Desenvolvimento de um Modelo de Gerenciamento de Redes de Telecomunicações Utilizando a Plataforma CORBA

Junior Toshiharu Saito

Dissertação de Mestrado

Instituto de Computação

Universidade Estadual de Campinas

Desenvolvimento de um Modelo de Gerenciamento de Redes de Telecomunicações Utilizando a Plataforma CORBA

Junior Toshiharu Saito[1]

Setembro de 2001.

Banca Examinadora

- Prof. Dr. Edmundo Roberto Mauro Madeira (Orientador)
- Prof. Dr. José Marcos Silva Nogueira
  Departamento de Ciência da Computação (DCC – UFMG)
- Prof. Dr. Nélson Luís Saldanha da Fonseca
  Instituto de Computação (IC – UNICAMP)
- Profa. Dra. Maria Beatriz F. de Toledo
  Instituto de Computação (IC – UNICAMP)

[1] Financiado pelo CNPq

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Saito, Junior Toshiharu

Sa28d Desenvolvimento de um modelo de gerenciamento de redes de telecomunicações utilizando a plataforma CORBA / Junior Toshiharu Saito -- Campinas, [S.P. :s.n.], 2001.

Orientador : Edmundo Roberto Mauro Madeira

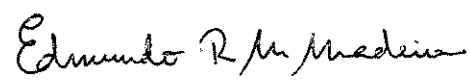
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. CORBA (Linguagem de programação de computador). 2. Programação orientada a objetos (Computação). I. Madeira, Edmundo Roberto Mauro. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Desenvolvimento de um Modelo de Gerenciamento de Redes de Telecomunicações Utilizando a Plataforma CORBA

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Junior Toshiharu Saito e aprovada pela Banca Examinadora.

Campinas, 14 de setembro de 2001.

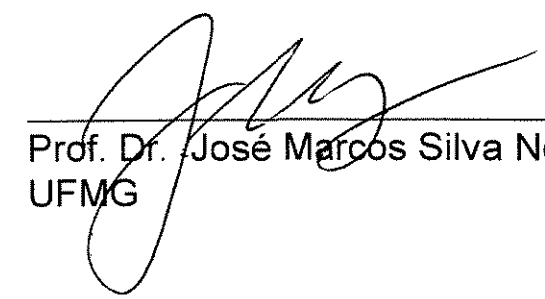


Edmundo Roberto Mauro Madeira

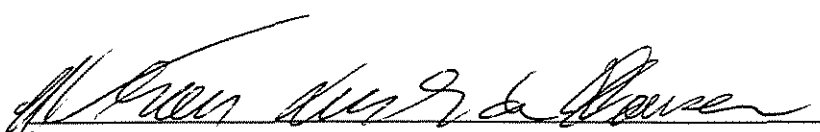
Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 14 de setembro de 2001, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. José Marcos Silva Nogueira
UFMG



Prof. Dr. Nelson Luís Saldanha da Fonseca
IC - UNICAMP

Prof. Dr. Edmundo Roberto Mauro Madeira
IC - UNICAMP

Agradecimentos

Primeiramente, agradeço a Deus, pois sem Ele, nada disso seria possível.

Aos meus pais e meus irmãos que me deram a maior força e apoio para continuar e nunca desistir.

Ao meu orientador Edmundo Madeira que me orientou para o desenvolvimento deste trabalho. Aos professores da pós-graduação e de graduação que me deram o conhecimento que tenho hoje.

Aos meus amigos da República – Luciano, Danillo, Gérson, Uber e Sandro – que sempre incentivaram e deram força nos momentos de fraqueza, esperança nos momentos de desespero e motivos para rir durante os momentos mais difíceis.

Aos meus amigos da Seicho-No-Ie de Campinas – Raquel Taira, Erton, Raquel Satto, Henrique, Jonady, Romeu, Akio, Sérgio, Marico, Reinaldo, Tioshi – pela amizade e pelos momentos de alegria. Ao meu amigo Mário Nakazato, que ajudou em meus estudos. Aos meus amigos de Maringá que sempre estiveram ao meu lado.

Ao engenheiro Ubirajara Seyr Júnior, pela sua ajuda. Aos verdadeiros amigos que sempre torceram por mim e que sempre estiveram ao meu lado.

Resumo

O gerenciamento de rede é uma tarefa muito importante para o funcionamento de uma rede, principalmente as redes de telecomunicações. A causa disto é o aumento do tamanho e da complexidade das redes que dificultam a deteção de falhas e baixo desempenho.

Outro fator de importância na gerência é permitir que este seja feito de forma descentralizada. O grupo OMG, analisando a possibilidade de se utilizar a sua plataforma CORBA para permitir esta forma de gerenciamento, lançou um conjunto de serviços para auxiliar a construção de aplicações para o gerenciamento de redes de telecomunicações.

Neste trabalho será apresentada uma arquitetura para o gerenciamento de redes de telecomunicações que utiliza objetos distribuídos. Esta arquitetura utiliza-se dos recursos existentes no Serviço de Notificação CORBA, várias ferramentas foram desenvolvidas.

Abstract

The network management is a task very important to its operation, 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.

Other important fact in network management is the decentralization of the managers, so in case of faults there will be a manager receiving the events. The CORBA architecture allows the decentralized network management, using the CORBA services.

In this dissertation an architecture to the management of telecommunication networks using distributed objects is presented. This architecture uses the existent resources in the CORBA Notification Service, many tools were developed.

Índice

Índice de Figuras

Índice de Tabelas

Introdução

As redes de telecomunicações possuem importância vital para o mundo atual. A partir dela, o mundo está interligado, podendo ser possível conversar ou transmitir dados com qualquer parte do mundo.

Para que tudo isso ocorra, existe um grande número de equipamentos interligados em funcionamento. A ocorrência de uma falha em algum desses equipamentos pode paralisar a rede, ocasionando prejuízos tanto para as operadoras quanto para os clientes. Devido ao tamanho das redes, a detecção de falhas não é uma tarefa simples, que pode ser feita manualmente. Há a necessidade de se desenvolver ferramentas para auxiliar no gerenciamento das redes de telecomunicações.

O gerenciamento de redes de telecomunicações é uma atividade vital para o funcionamento das redes. Entre as tarefas que constituem o gerenciamento estão a detecção de falhas e baixo desempenho, e o monitoramento dos serviços prestados e dos equipamentos existentes na rede.

Devido ao grande número de fabricantes, tipos de equipamentos e de diferentes tecnologias existentes, houve a necessidade de se definir um padrão para permitir a interoperabilidade entre os sistemas existentes. Para isso, a ITU-T definiu a arquitetura TMN [30], que fornece os pontos para permitir a interoperabilidade entre os equipamentos.

Entretanto o desenvolvimento de aplicações para gerência TMN mostrou ser uma tarefa complexa e de elevado custo, devido à complexidade do problema e ao ambiente diversificado onde as aplicações devem ser integradas.

O OMG (*Object Management Group*), grupo composto por desenvolvedores e empresas, incluindo empresas de telecomunicações, desenvolveu uma plataforma para ambientes distribuídos – CORBA [17] – onde vários domínios podem ser atendidos, entre eles, o domínio de telecomunicações. Para este domínio, foram apresentados vários serviços para atuar na gerência de redes sobre a plataforma: o Serviço de Eventos, o Serviço de Notificação e o Domínio de Eventos. O Serviço de Notificação foi desenvolvido para corrigir algumas deficiências existentes no Serviço de Eventos. Já o Domínio de Eventos é uma extensão do Serviço de Notificação, fornecendo novas funcionalidades.

Estes serviços oferecem facilidades para diminuir a complexidade de integração entre as aplicações.

A proposta deste trabalho é apresentar um modelo que utiliza o Serviço de Notificação e o Domínio de Eventos para o gerenciamento de redes de telecomunicações. A arquitetura CORBA fornece a facilidade de gerenciar diretamente através da interface do objeto gerenciado, principalmente o gerenciamento de serviços, comuns em redes de telecomunicações. O modelo utilizado para mostrar a utilização da arquitetura CORBA ainda tem como característica a escalabilidade, sendo capaz de se aumentar o número de equipamentos ou centrais ligadas a rede de telecomunicações, sem alterar o sistema de gerenciamento. Este modelo mostrará a facilidade de integrar as aplicações utilizando estes serviços. Devido ao grande número de funcionalidades existentes para o gerenciamento de redes, apenas algumas serão utilizadas, como por exemplo, a correlação de eventos e o log.

O trabalho está dividido em quatro capítulos. O segundo capítulo descreve os conceitos básicos utilizados neste trabalho, como CORBA, o gerenciamento de redes de telecomunicações e o gerenciamento de redes utilizando CORBA. O terceiro capítulo apresenta o modelo proposto e a descrição dos módulos que o compõe. O quarto capítulo descreve como foram implementados os módulos do modelo, os testes realizados e os resultados obtidos. E o quinto capítulo apresenta a conclusão deste trabalho.

Capítulo 2

Conceitos Básicos

Neste capítulo serão descritos os conceitos utilizados para o desenvolvimento deste trabalho. Estes conceitos são CORBA, gerenciamento de redes de telecomunicações, gerenciamento de redes utilizando CORBA e correlação de eventos.

2.1. CORBA

CORBA (*Common Object Request Broker Architecture*) [17] é a plataforma definida pelo OMG (*Object Management Group*) para permitir a interoperabilidade e a portabilidade entre as várias aplicações desenvolvidas em diferentes linguagens de programação e utilizadas em diferentes sistemas operacionais. A estrutura desta plataforma é descrita a seguir.

O OMG publicou a OMA (*Object Management Architecture*), que define os principais componentes presentes na arquitetura para computação distribuída: o ORB (cuja arquitetura é a CORBA), os serviços CORBA, as facilidades CORBA, as interfaces de domínio e os objetos da aplicação, como mostrado na Figura 1.

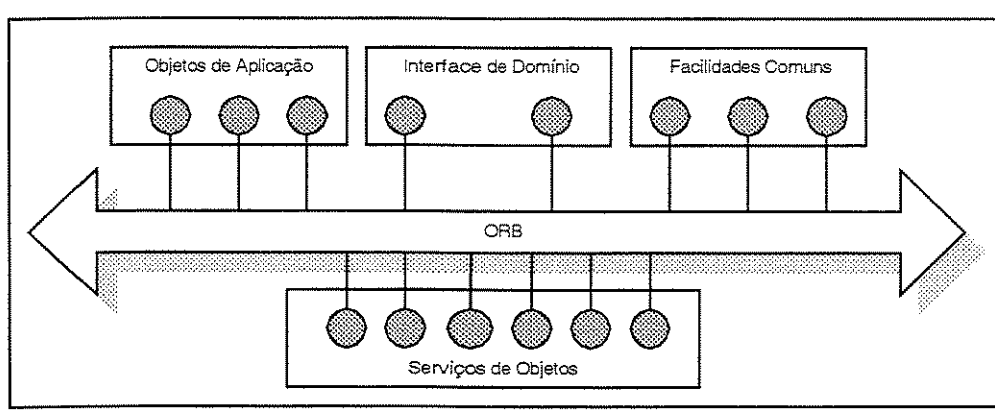


Figura 1: Componentes da Arquitetura CORBA.

2.1.1. ORB

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. A arquitetura do ORB é apresentada na Figura 2.

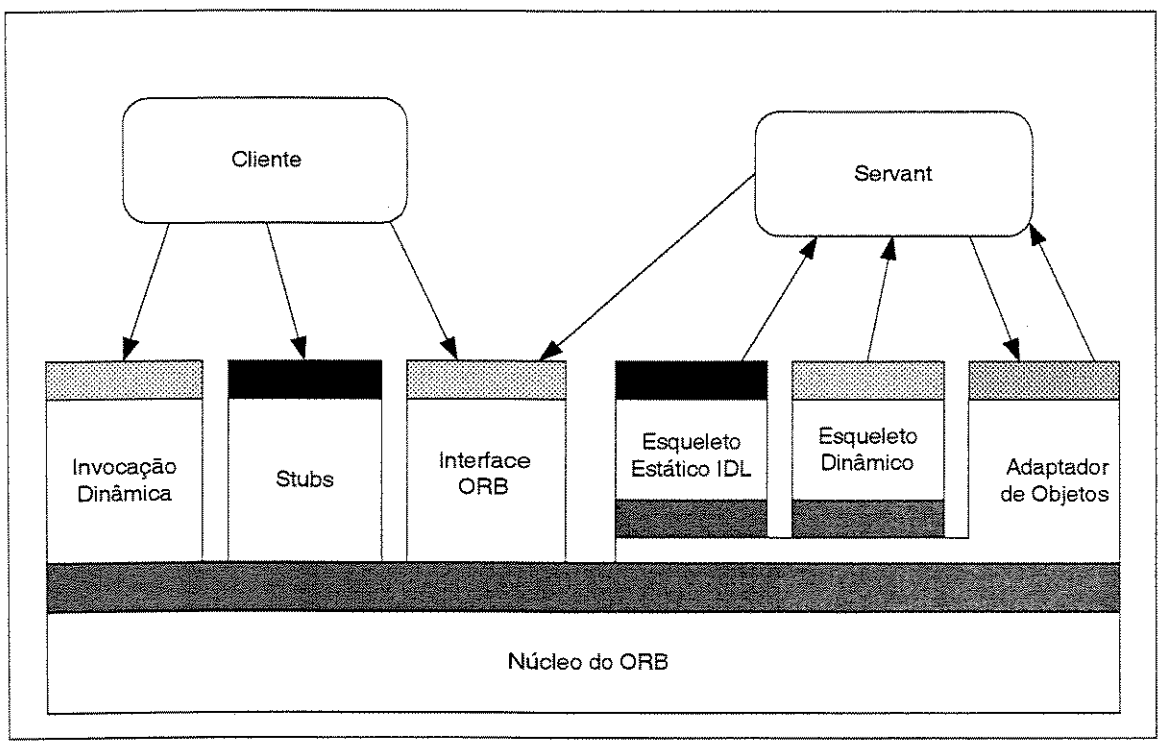


Figura 2: Arquitetura do ORB.

Como pode ser observado na Figura 2, existem o cliente e o servidor (*servant*) que se comunicam entre si utilizando o ORB. Esta comunicação é viabilizada pela utilização de interfaces. As interfaces definem os métodos e os atributos que estarão disponíveis aos clientes e que serão implementados pelos servidores. O OMG definiu uma linguagem padrão para a especificação de interfaces, o OMG IDL (*Interface Definition Language*). A partir destas definições são gerados os mapeamentos para as linguagens de programação. Após utilizar um compilador IDL, os *stubs* e os esqueletos são gerados. Um *stub* é um mapeamento estático utilizado pelo cliente para invocar um método de um objeto. Já o

esqueleto é o mapeamento estático utilizado pelo servidor para responder aos métodos invocados. Também é possível fazer uma invocação dinâmica, descobrindo as interfaces em tempo de execução.

Assim, quando o lado do cliente invoca um serviço de um objeto, pode fazer através de uma invocação dinâmica ou de uma invocação estática. A invocação estática é feita diretamente através das interfaces do objeto, o cliente então inicia a requisição através de chamadas às rotinas do *stub*. Já uma invocação dinâmica é feita em tempo de execução, obtendo a interface do objeto utilizando o repositório de interfaces.

Do lado do servidor, existem os *servants*. Um *servant* é um objeto ou uma entidade que implementa pedidos em um ou mais objetos. Os pedidos feitos por um objeto são transformados pelo ORB em invocações para um determinado *servant*. Além do *servant*, existe o gerente de *servants* (*servant manager*) que é um objeto associado a um POA (*Portable Object Adapter*). O POA é um adaptador de objetos que pode ser utilizado por múltiplos ORB's. O ORB invoca operações nos gerentes para ativar os *servants*. Um pedido pode ser atendido de forma estática ou dinâmica. Na forma estática, o mecanismo utilizado é o esqueleto IDL, gerado em tempo de compilação. E na forma dinâmica, é utilizado o esqueleto dinâmico que gera um esqueleto IDL com as informações necessárias (objeto de referência, um método e uma lista de parâmetros).

A interface ORB é utilizada por ambos, cliente e servidor, para acessar serviços gerais, como a iniciação do ORB. O núcleo do ORB é o componente responsável em fazer a comunicação entre o cliente e o objeto.

2.1.2. Objetos de Aplicação e Interfaces de Domínios

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 exemplo de domínios tem-se: Transporte, Financeiro, Comércio Eletrônico, Saúde e Telecomunicações. O Serviço de Notificação e o Domínio de Eventos se encontram no domínio das Telecomunicações.

2.1.3. Facilidades Comuns

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, mobilidade de agentes, correio eletrônico, entre outras.

2.1.4. Serviços de Objetos

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.

Entre os serviços existentes estão os seguintes: nomes (*Naming*), eventos (*Events*), ciclo de vida (*LifeCycle*), persistência (*Persistent Object*), transações (*Transactions*), controle de concorrência (*Concurrency Control*), relações (*Relationships*), propriedades (*Property*), segurança (*Security*), tempo (*Time*) e coleção (*Collections*).

2.2. Gerenciamento de Redes de Telecomunicações

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 a expansão da rede, gerenciar os recursos e controlar o acesso à rede.

Com o grande número de fabricantes, equipamentos e tecnologias para as redes de telecomunicações, surgiu a necessidade de um padrão para auxiliar no gerenciamento das redes de telecomunicações. Para isso, a ITU apresentou o TMN (*Telecommunications Management Network*) [30] para padronizar a gerência das redes de telecomunicações. O TMN apresenta uma arquitetura para permitir a interconectividade e a comunicação entre os diferentes sistemas que podem existir na rede. A seguir, as partes que compõem o TMN são descritas.

2.2.1. A Arquitetura TMN

A arquitetura fornecida pelo TMN foi desenvolvida para permitir flexibilidade, confiabilidade, interoperabilidade, escalabilidade e facilidade para implementá-la. Esta

arquitetura é incorporada nas redes de telecomunicações, permitindo receber e mandar informações e gerenciar os recursos existentes na rede.

Esta arquitetura é definida na série de especificações M.3000 da ITU. O padrão inclui os seguintes elementos:

- *Common Management Information Protocol* (CMIP): define o protocolo do serviço de troca de mensagens entre as entidades
- *Guideline for Definition of Managed Objects* (GDMO): fornece um template para classificação e descrição dos recursos gerenciados
- *Abstract Syntax Notation One* (ASN.1): define a sintaxe das regras para os tipos de dados
- Modelo de referência para interconexão de sistemas abertos: define as sete camadas do modelo OSI.

A arquitetura TMN pode ser descrita de três formas: arquitetura funcional, arquitetura lógica e arquitetura de informação. Para este trabalho serão descritas apenas as duas primeiras.

2.2.2. Arquitetura Funcional

O TMN permite 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.

O TMN é representado por vários blocos funcionais, os quais são apresentados na Figura 3.

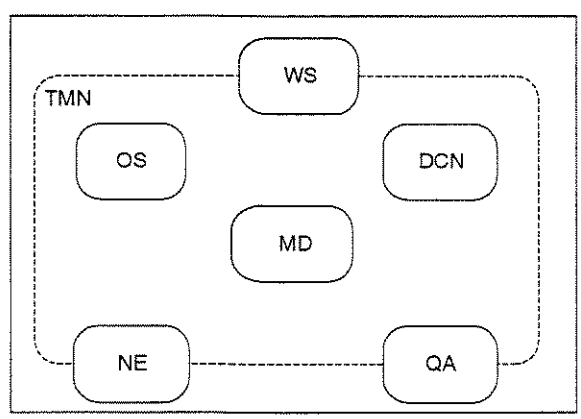


Figura 3: Blocos do TMN.

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. Representa as camadas 1,2 e 3 do modelo OSI.
- **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.
- **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.
- **NE** (*Network Element*): possui 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.

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.

2.2.2.1 Interfaces Padrões

A arquitetura TMN apresenta interfaces padrões para permitir a comunicação entre dois blocos, como exemplo um OS com um MD. Estas interfaces definem funções que executam as tarefas de gerenciamento. As interfaces são as seguintes:

- **Q**: esta interface está presente entre dois blocos TMN dentro de um mesmo domínio TMN. A interface Qx possui informação que é compartilhada entre o MD e o NE. A interface Qx existe entre NE e MD; QA e MD, e MD e MD. A interface Q3 é a interface do OS. Qualquer componente que se conecta ao OS utiliza a interface Q3. A interface Q3 está entre NE e OS, QA e OS, MD e OS, e OS e OS.
- **F**: a interface F está entre uma WS e o OS, e entre o WS e o MD.
- **X**: a interface X existe entre dois OS's em domínios diferentes, ou entre um OS que suporte TMN e outro OS que não suporte.

Além destas interfaces, existem dois pontos de referência que estão fora do escopo do TMN: g e m. Os pontos de referência definem as fronteiras de serviços entre dois blocos funcionais de gerenciamento. O ponto de referência g está entre duas entidades que não suportam TMN, e o ponto de referência m está na parte do WSF que não suporta TMN e QAF. As interfaces entre os componentes TMN são mostradas na Figura 4.

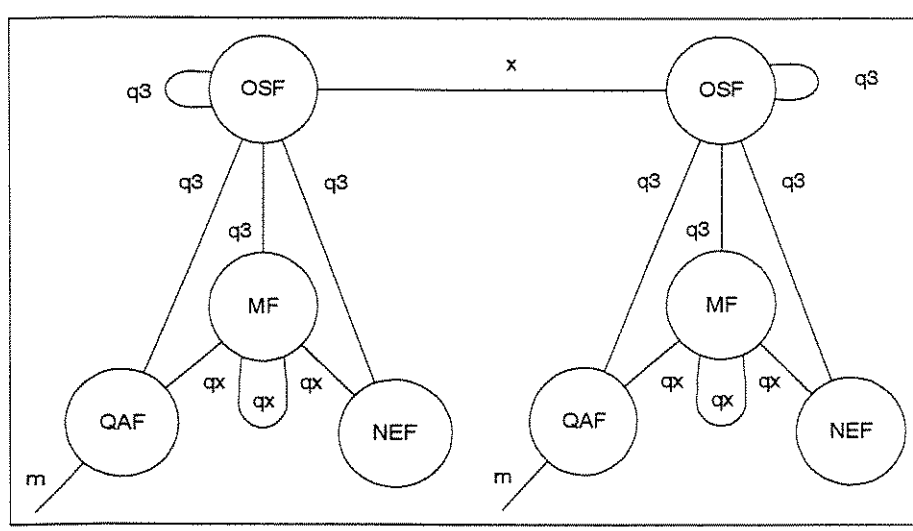


Figura 4: Interfaces entre os Componentes TMN.

2.2.3. Arquitetura Lógica

A arquitetura lógica fornecida pelo TMN é composta por camadas que definem o nível de gerenciamento que se deseja obter. O mesmo tipo de funções pode ser implementado nos vários níveis, do mais alto, que gerencia a corporação ou as metas a

serem atingidas, ao nível mais baixo, que é definido pela rede ou pelos recursos da rede. A partir da camada mais alta para a mais baixa temos as seguintes camadas:

- **Camada de gerenciamento de negócios** (BML – *business-management layer*): É o planejamento de alto-nível, orçamento, definição de metas, decisões executivas, acordos em nível de negócios.
- **Camada de gerenciamento de serviços** (SML – *service-management layer*): gerencia os serviços fornecidos aos usuários, usando a informação presente na NML. A SML é o ponto de interação entre os provedores de serviços e outros domínios administrativos.
- **Camada de gerenciamento de rede** (NML – *network management layer*): fornece a visão da rede inteira, baseado nas informações dos NE's presentes no EML. O NML pode gerenciar um NE individualmente ou todos os NE's como um grupo. Todas as atividades da rede são coordenadas pelo NML.
- **Camada de gerenciamento de elemento** (EML – *element-management layer*): gerencia cada elemento da rede, pode possuir elementos gerenciadores. Em geral, um elemento gerenciador é responsável por um subconjunto de NE's. Um elemento gerenciador gerencia os dados de um elemento da rede, logs e atividades.
- **Camada de elementos da rede** (NEL – *network-element layer*): apresenta a informação gerenciada como um NE individual.

2.3. Gerenciamento de Redes utilizando CORBA

O OMG desenvolveu o Serviço de Eventos [15], um serviço geral que permitiu a troca de mensagens utilizando eventos, de forma assíncrona. Entretanto este serviço apresentou algumas deficiências quanto à sua utilização na área de Telecomunicações. Para corrigir essas deficiências, foi desenvolvida uma extensão, o Serviço de Notificação [16].

O Serviço de Notificação apresenta algumas vantagens se comparado com os sistemas que utilizam CMIP ou SNMP. A principal vantagem da utilização de CORBA é a ausência de um agente para enviar os eventos, tarefa que pode ser feita pela própria interface do objeto gerenciado. Isso facilita a gerência de serviços. A Figura 5 mostra as possibilidades de utilização do Serviço de Notificação para o gerenciamento.

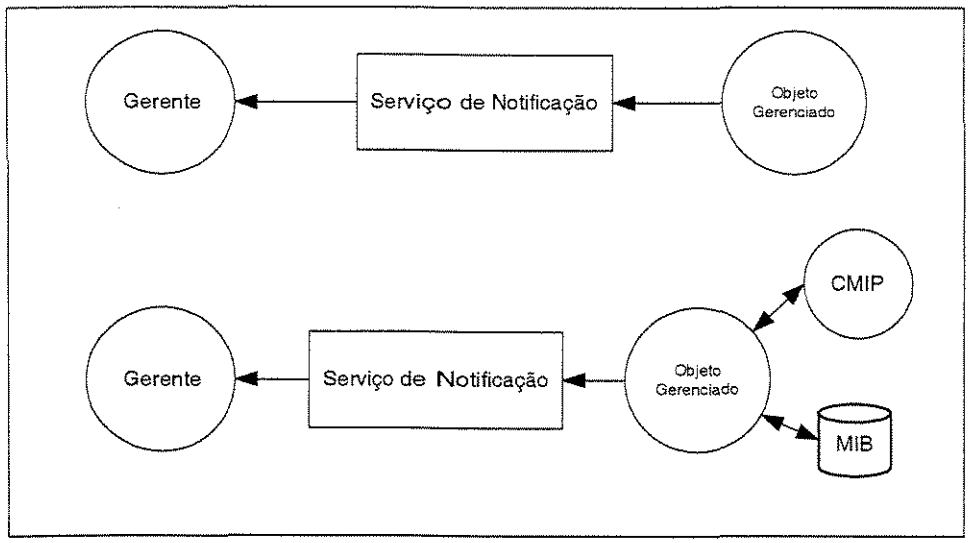


Figura 5: Possibilidades de Utilização do Serviço de Notificação.

2.3.1. Serviço de Notificação

O Serviço de Notificação é uma extensão do Serviço de Eventos, por isso praticamente todas as características foram mantidas, sendo acrescentadas apenas novas funcionalidades. Os Serviços de Notificação e o de Eventos permitem a comunicação assíncrona entre cliente e servidor. Estes serviços são compostos por aplicações clientes, os consumidores, que processam os eventos; e as aplicações servidoras, os produtores, que geram os eventos. A comunicação entre as aplicações é feita através de canais, cuja finalidade é decompor a comunicação, desacoplando os produtores e os consumidores. Num canal é possível a conexão de vários consumidores e vários produtores.

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, mostradas na Tabela 1.

Tabela 1: Possibilidades de Comunicação entre Produtor, Canal de Eventos e Consumidor.

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

Para que o canal de evento 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 6. Para criar os *proxies* existem os objetos de administração (Admin), que 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. 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.

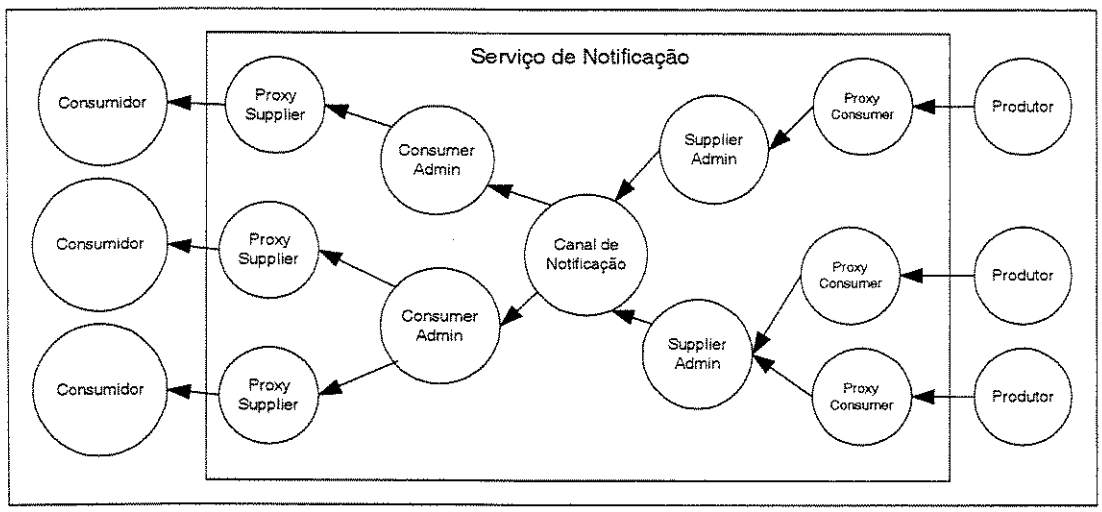


Figura 6: Produtor, Consumidor e seus Proxies.

Quanto aos eventos, o Serviço de Eventos especifica dois tipos de eventos:

- *Any:* envolve a transmissão de eventos genéricos. É de fácil utilização, mas muitas aplicações requerem a troca de eventos tipados.
- *Typed:* tipado, os eventos do tipo *Typed* são eventos que 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 Serviço de Notificação suporta os dois tipos de eventos existentes no Serviço de Eventos e acrescenta um novo tipo de evento: o estruturado. O evento estruturado foi desenvolvido devido à dificuldade de se utilizar o evento *Typed*. A estrutura do evento estruturado é apresentada na Figura 7.

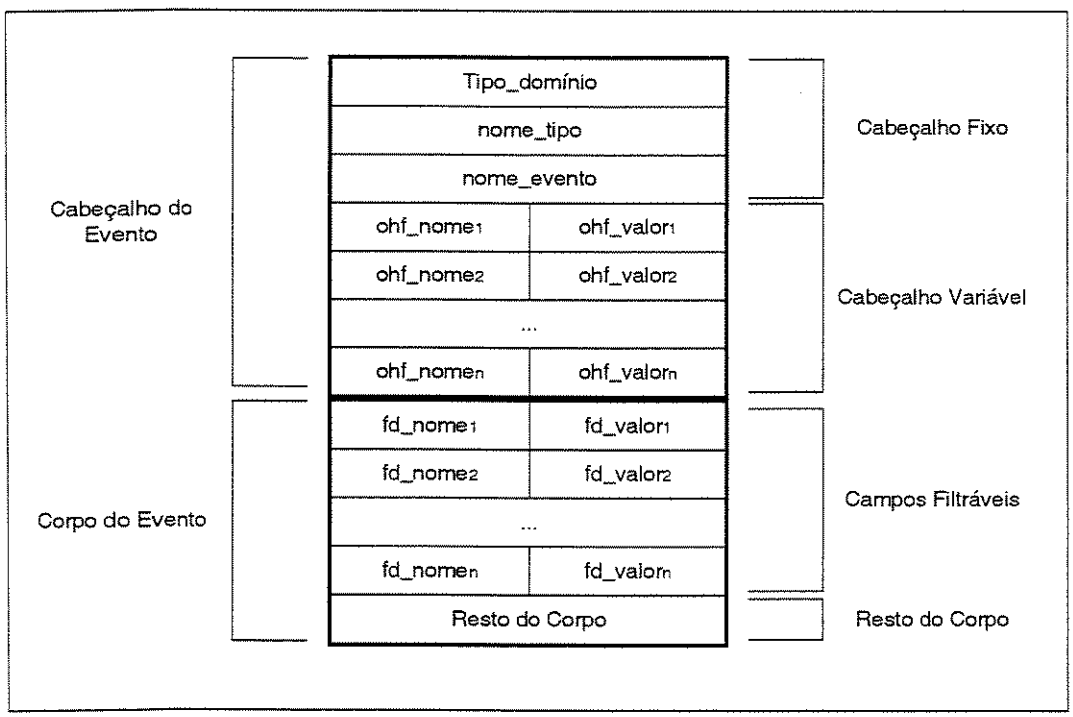


Figura 7: Estrutura do Evento Estruturado.

Um evento estruturado é 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 tipo_domínio, nome_tipo e nome_evento; e o

cabeçalho variável composto por tuplas do tipo nome-valor que definem atributos como prioridade, tempo de vida 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, onde o valor é armazenado num campo do tipo *Any*; e o resto do corpo composto pelo campo resto_do_corpo que pode ser utilizado para armazenar qualquer tipo de informação adicional.

A Figura 8 apresenta dois exemplos de eventos estruturados. A estrutura de ambos os eventos é a mesma, com a mesma quantidade de campos. O campo tipo_domínio de ambos os eventos é do tipo *Telecommunication*, o campo nome_tipo possui os valores *CommunicationAlarm* para o evento (a) e *Equipment*, para o (b). Para o campo nome_evento, o primeiro evento é o *Local Node Transmission Error*, e o segundo, *Equipment MalFunction*. Quanto ao campo variável, este possui apenas dois campos: prioridade e *Timeout*. Já para corpo de campos filtráveis, foram utilizados cinco campos: *ManagedObjectClass*, *ManagedInstanceClass*, *EventTime*, *ProbableCause* e *PerceivedSeverity*.

Telecommunication	
CommunicationsAlarm	
Local node transmission error	
Prioridade	1
Timeout	100
ManagedObjectClass	multiplexer
ManagedObjectInstance	eqp01
EventTime	2001123135959
ProbableCause	power problem
PerceivedSeverity	Critical
Power Supply Failure	

(a)

Telecommunication	
Equipment	
Equipment malfunction	
Prioridade	1
Timeout	100
ManagedObjectClass	exchange
ManagedObjectInstance	exchg003
EventTime	2032124167959
ProbableCause	Comm problem
PerceivedSeverity	Critical
Communication Failure	

(b)

Figura 8: Exemplos de Eventos Estruturados.

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 9. Existem dois tipos de filtros: os filtros gerais (*filter object*) e os filtros de mapeamento (*mapping filter object*).

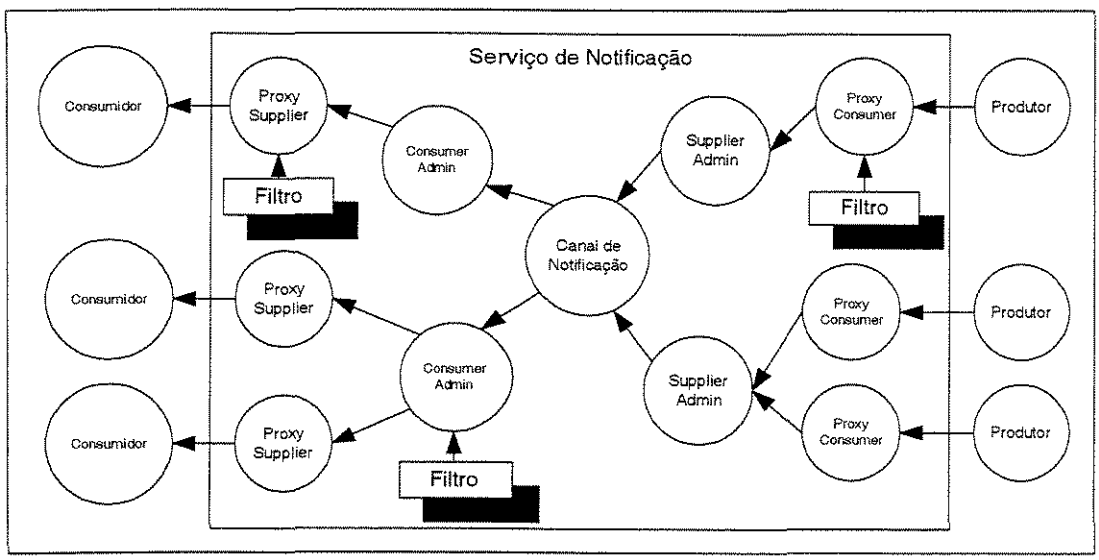


Figura 9: Filtros.

O filtro (*filter object*) é o responsável em tomar as decisões de envio do evento nos proxies, este tipo de filtro possui um conjunto de restrições. Cada restrição é uma estrutura de dados composta de duas partes: uma seqüência de estruturas de dados, cada uma indicando um tipo de evento (domínio, tipo), e uma expressão booleana que indica a restrição, como apresentado na Figura 10. Neste exemplo, tem-se um filtro que aceita apenas eventos cujo domínio é *Telecommunication* e o tipo é *CommunicationsAlarm*.

```
"($domain_type == "Telecommunication" and $type_name == "CommunicationsAlarm")"
```

Figura 10: Expressão para um Filtro.

O outro filtro, o filtro de mapeamento, define como o *proxy* tratará os eventos com certos valores de qualidade de serviço. Este filtro pode alterar duas propriedades presentes no evento: a prioridade e o tempo de vida.

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 ao 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.

O Serviço de Notificação conta ainda com um repositório de eventos que pode ser utilizado para especificar as restrições dos filtros.

Outro recurso importante 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.

2.3.2. Domínio de Eventos

Novamente, um novo serviço foi desenvolvido tendo como base o Serviço de Notificação, o domínio de eventos (*Event Domain*) [18, 19]. Esta extensão do Serviço de Notificação adiciona algumas funcionalidades não presentes no serviço, 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.
- Configurar a qualidade de serviço de todos os canais com uma operação.

No Serviço de Notificação, um produtor conecta a um *proxyconsumer* fornecido em um canal de eventos, e o consumidor conecta a um *proxysupplier*. Quando este canal de eventos conecta a um outro canal como um produtor, cria-se um objeto *proxysupplier*, e este se conecta a um *proxyconsumer* fornecido pelo canal destinatário. Este *proxyconsumer*, por sua vez, se conecta com o *proxysupplier* do canal produtor. Como resultado ambos os canais estão conectados.

Embora seja possível formar uma conexão entre canais usando as funções do Serviço de Notificação, o processo de conexão pode envolver um complicado procedimento indicado abaixo, com o aumento nos passos na ordem quadrática do número de canais:

- Pegue uma referência para o *SupplierAdmin* de um canal de eventos, o produtor de eventos.
- Pegue uma referência para o *ProxyConsumer* do *SupplierAdmin*.
- Pegue uma referência para o *ConsumerAdmin* do outro canal de eventos, o consumidor.
- Pegue uma referência para o *ProxySupplier* do *ConsumerAdmin*.
- Conecte o *ProxyConsumer*, usando a referência do *ProxySupplier* obtida como parâmetro de entrada.
- Similarmente, conecte o *ProxySupplier* usando a referência do *ProxyConsumer* como parâmetro de entrada.
- Use este processo de conexão para gerenciar as referências de *proxy*.

Todo esse processo resulta na conexão entre dois canais, onde um assume o papel de produtor e o outro, consumidor. Por assumir essa configuração, a transmissão de eventos é feita de forma direta. A Figura 11 apresenta a conexão entre estes dois canais.

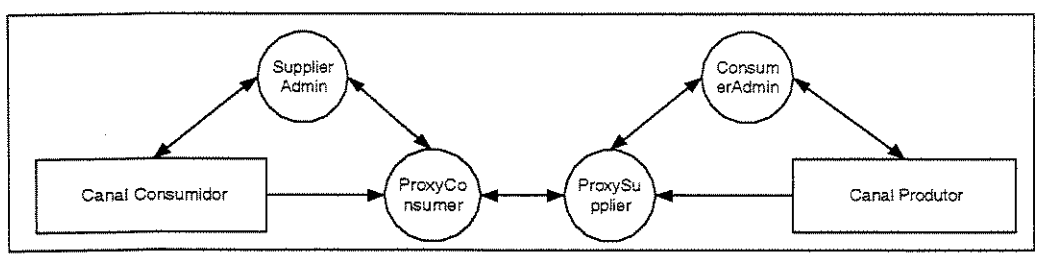


Figura 11: Conexão entre dois Canais.

Utilizando o Domínio de Eventos, esta tarefa é realizada com apenas uma operação, resumindo-se à criação de uma conexão entre dois canais. Uma interface *EventDomain* pode ser definida por manipulação dos passos acima em uma operação, assim como a gerência das instâncias de *proxy* e status da conexão, e fornecendo uma interface para responder a consultas. A partir desta interface, um canal de eventos é registrado na rede,

que retorna um inteiro como um *channelId*. Após isto, este identificador é usado quando estabelecer, checar ou remover conexões entre canais.

Quanto à transmissão de eventos, isto é feito de forma direta, uma vez que os canais são conectados entre si como produtores ou consumidores, os eventos trafegam dos produtores para os consumidores.

2.3.3. Serviço de Log

A arquitetura CORBA inclui um Serviço de Log [20] para a área de Telecomunicações. Este serviço oferece suporte total para a especificação X.735 [3]. O Serviço de Log tem como base para a sua implementação o Serviço de Notificação. Desta forma, o log funciona como um canal de Notificação oferecendo várias vantagens para o sistema:

- O Log oferece suporte para múltiplos produtores e múltiplos consumidores.
- O Log pode possuir um filtro para selecionar os eventos de seu interesse.
- Existe a possibilidade dos logs formarem uma rede, da mesma forma que os canais de notificação.

2.4. Correlação de Eventos

A correlação [11, 12, 13, 14, 26] é uma técnica 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.

A correlação pode ser feita em diversos níveis de configuração, incluindo desde o nível de elemento de rede individual, até o nível máximo, que envolve toda a rede.

Em nível mais baixo, constitui-se de processos mais simples e, conseqüentemente, mais rápidos. Não leva em consideração o contexto mais amplo, por isso sofre uma acentuada miopia, que impede de detectar possíveis reflexos que problemas locais podem ocasionar na rede como um todo.

No nível mais alto, todas as informações relevantes podem, em teoria, ser oferecidas ao mecanismo de correlação, que desta forma tem uma ampla visão do sistema gerenciado e pode diagnosticar problemas através dos seus reflexos sobre a rede como um todo. Mas a

grande quantidade de informações disponíveis provoca um aumento da complexidade do problema, que muitas vezes torna-se intratável. A Seção seguinte apresenta alguns tipos de correlação existentes, métodos e algoritmos.

2.4.1. Tipos de Correlação

Vários tipos de correlação podem ser identificados baseados nas operações executadas sobre os alarmes disponíveis. A seguir são descritos os mais importantes.

2.4.1.1 Compressão

Consiste em detectar, a partir da observação dos alarmes recebidos em uma dada janela de tempo, múltiplas ocorrências de um mesmo evento, substituindo os alarmes correspondentes por um único alarme, possivelmente indicando quantas vezes o evento ocorreu durante o período de observação.

2.4.1.2 Supressão Seletiva

É a inibição temporária dos alarmes referentes a um dado evento, segundo critérios continuamente avaliados pelo sistema de correlação, relacionados ao contexto dinâmico do processo de gerência de rede. Os critérios de supressão geralmente estão vinculados à presença de outros alarmes, ao relacionamento temporal entre alarmes ou a prioridades estabelecidas pelos gerentes da rede.

2.4.1.3 Filtragem

Consiste em suprimir um determinado alarme, em função dos valores de um conjunto de parâmetros, previamente especificados. Em um sentido restrito, a filtragem leva em consideração apenas os parâmetros do alarme que estiver sendo filtrado. Em um sentido mais amplo, a filtragem pode levar em consideração quaisquer outros critérios. Nesse caso, que poderia ser caracterizado como uma filtragem inteligente, o conceito de filtragem se expande, podendo englobar diversos outros tipos de operações, tais como compressão.

2.4.1.4 Contagem

Consiste em gerar um novo alarme a cada vez que o número de ocorrências de um determinado tipo de evento ultrapassar um limiar previamente estabelecido.

2.4.1.5 Escalação

É uma operação na qual, em função do contexto operacional, um alarme é suprimido, sendo criado em seu lugar um outro alarme, no qual um parâmetro (p. ex., o parâmetro severidade) assume um valor mais alto. O contexto operacional inclui, dentre outros fatores, a presença de outros alarmes, o relacionamento temporal entre alarmes, o número de ocorrências de um evento em uma dada "janela" de tempo e as prioridades estabelecidas pelos gerentes da rede.

2.4.1.6 Generalização

Consiste em substituir um alarme, em função do contexto operacional, pelo alarme correspondente à sua superclasse. Como exemplo, na ocorrência simultânea de alarmes correspondentes a todas as rotas que utilizam como meio físico um determinado cabo, cada um dos alarmes originais pode ser substituído por um alarme indicando defeito do cabo; em seguida, através de uma operação de compressão, todos os alarmes repetidos podem ser substituídos por um alarme único. Esta operação está baseada em raciocínio do tipo indutivo, que permite a ampliação do escopo do conhecimento, às custas do aumento da complexidade do problema e da introdução de um certo grau de incerteza no resultado da correlação.

Dois tipos principais de generalização podem ser identificados: generalização por simplificação de condições e generalização baseada em instâncias. No primeiro caso são ignoradas ou desprezadas uma ou mais das condições definidas como necessárias à identificação do alarme de classe mais alta. No segundo caso, um novo alarme pode ser gerado a partir da associação das informações correspondentes a dois ou mais alarmes recebidos.

2.4.1.7 Especialização

É a operação inversa à generalização, que consiste em substituir um alarme por um outro, correspondente a uma subclasse. Esta operação, baseada em raciocínio do tipo dedutivo, não acrescenta novas informações e relação às que já estavam implicitamente presentes nos alarmes originais e na base de dados de configuração, mas é útil no evidenciamento das consequências que um evento numa determinada camada de gerência pode ocasionar nas camadas de gerência superiores.

Como exemplo de uma possível especialização, o sistema de correlação pode gerar, sempre que uma determinada rota for interrompida, um alarme para cada um dos serviços afetados pela interrupção. Desta forma, através da especialização, estarão sendo evidenciadas as consequências de uma falha na camada de gerência de rede de telecomunicações sobre as entidades da camada de gerência de serviços de telecomunicações.

2.4.1.8 Relacionamento Temporal

É uma operação na qual o critério para correlação depende da ordem ou do tempo em que são gerados ou recebidos os alarmes. Diversas relações temporais podem ser definidas, utilizando conceitos tais como: Depois-de, em-seguida-a, antes-de, precede, enquanto, começa, termina, coincide-com e sobrepõe-se-a.

2.4.1.9 Aglutinação

Consiste na geração de um novo alarme a partir da verificação do atendimento, pelos alarmes recebidos, de padrões complexos de correlação. A operação de aglutinação também pode levar em consideração o resultado de outras correlações e o resultado de testes realizados na rede.

2.4.2. Métodos e Algoritmos

Nesta Seção, serão descritos os métodos e os algoritmos mais utilizados e estudados para a correlação de eventos. Algumas destas técnicas utilizam técnicas de inteligência artificial, enquanto outras tentam utilizar métodos *ad hoc*.

2.4.2.1 Correlação Baseada em Regras

Esta é uma das abordagens que utiliza a inteligência artificial. Neste método de correlação, o conhecimento geral sobre determinada área está representado em um conjunto de regras. Já o conhecimento específico é expresso através de asserções e armazenado em bancos de dados.

Uma regra consiste de duas expressões ligadas por um conectivo de implicação. O lado esquerdo de cada regra contém um pré-requisito que precisa ser satisfeito pelo banco de dados para que a regra seja aplicável. O lado direito possui a ação que deve ser executada se a regra for aplicada. A aplicação de uma regra altera o banco de dados.

Existem dois modos de operação:

- Direto (*forward*): partindo de um estado inicial, uma seqüência de passos é construída levando até a solução do problema.
- Reverso (*backward*): parte de uma configuração que corresponde à solução do problema e se constrói uma seqüência de passos que leva à configuração correspondente ao estado inicial.

2.4.2.2 Lógica Difusa

A lógica difusa é uma alternativa para lidar com a incerteza e a imprecisão que algumas aplicações de gerência de redes de telecomunicações contêm. A lógica difusa é composta pelo sistema lógico tradicional, os sistemas lógicos multivalorados, a teoria das probabilidades e a lógica probabilística.

2.4.2.3 Redes Bayesianas

Consiste na utilização de um grafo acíclico dirigido no qual cada nó representa uma variável aleatória à qual são associadas as probabilidades condicionais subjetivas, dadas todas as possíveis combinações de valores das variáveis representadas pelos nós predecessores diretos. Uma probabilidade subjetiva expressa o grau de crença de um especialista em relação à ocorrência de um dado evento, a partir das informações de que esta pessoa dispõe até o momento.

2.4.2.4 Raciocínio Baseado em Modelos

É um paradigma da área de inteligência artificial que tem diversas aplicações na correlação de alarmes. Consiste em representar um sistema através de um modelo estrutural e de um modelo funcional. No caso de um sistema de gerência de redes de telecomunicações, a representação estrutural inclui a descrição dos elementos de rede e da topologia.

2.4.2.5 Quadro Negro

Esta é uma técnica composta de um banco de dados global, o quadro negro, diversas fontes de conhecimento e um mecanismo de escalonamento. O quadro negro é responsável por armazenar os elementos de solução produzidos pelo sistema durante o processo de

resolução do problema. As fontes de conhecimento são processos responsáveis por gerar os elementos de solução e armazená-los no quadro negro.

2.4.2.6 Filtragem

Consiste na utilização de filtros que selecionam as notificações de alarmes a partir do pedido do operador, segundo critérios como área geográfica onde o alarme foi originado, área técnica ou grau de severidade do alarme. Apesar de conseguirem reduzir a quantidade de informações a serem exibidas, os critérios de corte desses filtros podem não facilitar a identificação das falhas que causaram a emissão das notificações de alarme.

2.4.2.7 *Event Forwarding Discriminator* – EFD

O *Event Forwarding Discriminator* consiste na utilização de um discriminador que determina quais os relatórios de eventos em potencial devem ser transferidos, sob a forma de relatórios de eventos, para um destino e durante um determinado intervalo de tempo especificado.

2.4.2.8 Raciocínio Baseado em Casos

O raciocínio baseado em casos é uma alternativa ao raciocínio baseado em regras. Nesta técnica, a unidade básica de conhecimento é um caso, consistindo de registros contendo os aspectos mais relevantes de episódios passados. Os casos, então, são armazenados, recuperados, adaptados e utilizados na solução de novos problemas.

2.4.2.9 Localização Explícita

A localização explícita propõe que cada alarme seja associado com uma informação sobre a localização da falha, consistindo de um conjunto que contém todas as localizações possíveis. Inicialmente é suposto que os alarmes sejam confiáveis e que há apenas uma falha na rede, assim a falha se localizará na intersecção dos conjuntos de localizações indicados pelos diversos alarmes. Em seguida, o modelo é estendido para cobrir múltiplas falhas, em um ambiente mais realístico onde os alarmes recebidos até podem não ser confiáveis.

2.4.2.10 Correlação por Votação

Nesta técnica, os alarmes não contêm votos para cada nó individual, mas para todos os nós de uma dada direção. Por isso, o sistema de correlação deve conhecer a topologia da

rede para que ao saber o número de votos de um alarme para uma dada direção, cada um dos nós daquela direção receba aquele número de votos. Em seguida, é feita a totalização dos votos de cada nó e a escolha do nó como possível localização das falhas.

2.4.2.11 Correlação Proativa

A intenção desta abordagem é identificar problemas em potencial antes mesmo que eles aconteçam. Isto é feito descobrindo padrões que caracterizam o comportamento atual e as tendências de comportamento futuro da rede. Para isso são utilizadas técnicas de *data mining* e de *knowledge discovery*.

2.4.2.12 Correlação Distribuída

Consiste em particionar a rede em diversos domínios estáticos, disjuntos e logicamente autônomos, cada um deles gerenciado por um único centro de gerência. Cada centro possui uma visão limitada do estado dos demais domínios. Entretanto, gerentes de diferentes domínios comunicam-se entre si e trocam informações sobre o estado de seus domínios.

2.4.2.13 Redes Neurais Artificiais

A idéia desta abordagem é utilizar uma rede neural artificial. Seu funcionamento é baseado no cérebro humano. A rede neural artificial é composta de "neurônios" (conceitualmente, cada neurônio é uma unidade de processamento autônoma) que se conectam entre si para fazer a troca de sinais. Uma rede neural artificial é capaz de "aprender" durante a operação do sistema, o que torna uma boa alternativa para a correlação de alarmes e diagnóstico de falhas, onde as relações entre falhas e alarmes nem sempre são bem definidas e os dados disponíveis, às vezes, ambíguos ou inconsistentes.

2.4.2.14 Diagnóstico por Comparação de Resultados de Testes

Consiste em fazer os nós da rede executarem uma série de tarefas conhecidas. O diagnóstico dos nós ou enlaces defeituosos é feito a partir das discrepâncias observadas nos resultados dos testes. A precisão do diagnóstico pode ser feita através do número de vezes que a tarefa é repetida.

2.4.2.15 Correlação por codificação

A idéia da correlação por codificação [27] é simples, para cada problema existe um conjunto de sintomas. Os eventos de sintomas podem ser eventos gerados por outros objetos ligados à fonte do problema. Assim, o conjunto de eventos ocasionados por um problema é tratado como um código que identifica o problema. A técnica então decodifica o código descobrindo a origem do problema. Os códigos são armazenados em um conjunto de códigos, o *codebook*.

Capítulo 3

Modelo Proposto para Gerenciamento de Redes de Telecomunicações

As redes de telecomunicações são compostas por centrais de comutação locais que atendem a um determinado número de clientes. As centrais se localizam em estações locais onde existem outros equipamentos necessários para seu funcionamento. As centrais são interligadas entre si, permitindo que clientes em centrais locais diferentes possam se comunicar [6]. O modelo de gerenciamento de redes de telecomunicações desenvolvido utiliza-se de uma pequena rede composta por várias estações locais, onde cada estação 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 e a comunicação entre as centrais. No caso do modelo de gerenciamento utilizando CORBA, a rede de comunicação de dados (DCN) é construída utilizando uma rede de canais de Notificação CORBA. Esta rede permite a escalabilidade do sistema por permitir que mais centrais ou equipamentos sejam conectados ao modelo sem a necessidade de operações complexas, bastando conectar-se a um dos canais ou criar um novo canal. O modelo 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, o modelo possui um correlacionador capaz de realizar a correlação de eventos, para determinar a origem dos eventos.

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

As redes de telecomunicação permitiram inicialmente a transmissão de voz através de longas distâncias. Hoje, a utilização de uma rede de telecomunicação não se limita apenas à transmissão de voz, podendo transmitir dados e vídeos.

Existem dois tipos de redes de telecomunicações: a fixa e a móvel. Uma rede de telecomunicações fixa é composta por terminais fixos 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 mostrada na Figura 12.

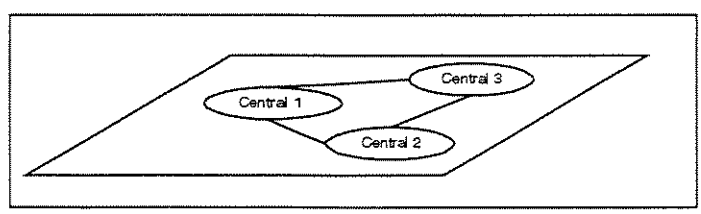


Figura 12: Representação de uma Rede de Telecomunicações.

A rede móvel é utilizada por equipamentos portáteis, como os telefones celulares. Esse tipo de comunicação é possível graças aos sinais de rádio utilizados para fazer a comunicação. A estrutura de uma rede móvel é formada por um grande número de rádios transmissores/receptores de baixa potência (denominados de estações rádio-base) dispostos numa determinada área geográfica. Cada estação opera numa área limitada chamada célula, como mostrado na Figura 13. Para diminuir a possibilidade de interferência entre os usuários, normalmente as freqüências são reutilizadas somente em células não adjacentes. O tamanho destas células não é uniforme, dependendo de fatores como números de usuários e relevo.

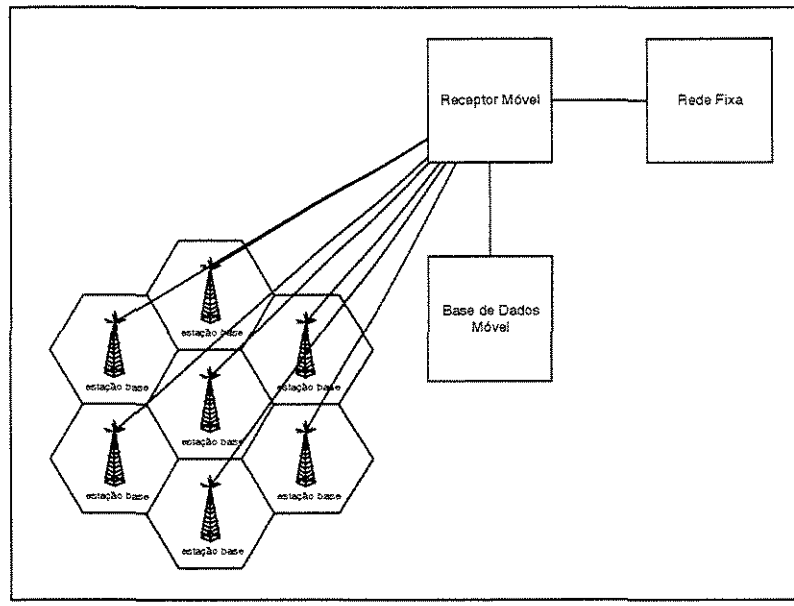


Figura 13: representação de uma área dividida em células.

O modelo proposto pode gerenciar tanto uma rede de telecomunicações fixa quanto uma rede de telecomunicações móvel. Para este trabalho, será considerada apenas a rede de telecomunicações fixa, onde as centrais possuem vários tipos de equipamentos entre eles: multiplexador, central de comutação 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. Para o caso de uma rede de telecomunicações móvel, a única diferença está em alguns equipamentos a serem gerenciados como as estações de rádio-base.

A ligação entre as centrais pode ser feita por vários tipos de meios: fios de cobre, cabos coaxiais, fibras ópticas, ondas de rádios e microondas. A ligação entre duas centrais é chamada de tronco.

Cada central possuirá seu próprio canal de notificação, por onde os agentes dos equipamentos enviam os alarmes que geram, como mostrado na Figura 14. A utilização de um canal por central facilita a escalabilidade do sistema, pois permite que mais equipamentos sejam conectados a esta central, sem a necessidade de alteração dos outros componentes do modelo. Os alarmes emitidos podem determinar a ocorrência de alguma falha em um dos troncos que interligam as centrais.

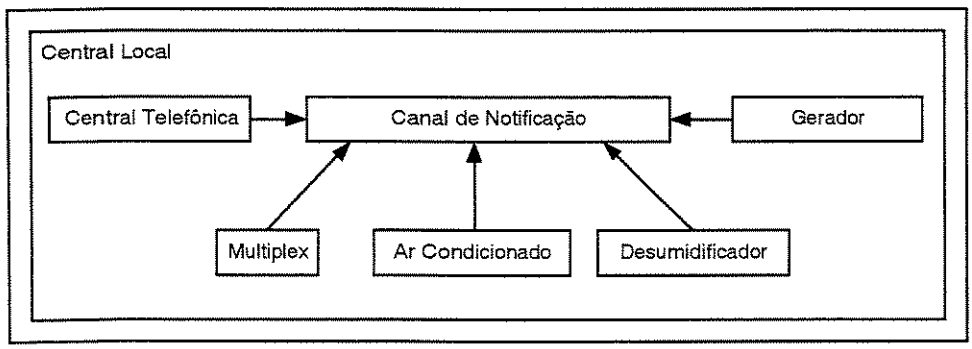


Figura 14: Central Telefônica.

3.2. Gerente

É o objeto responsável em receber os alarmes correlacionados ou não. O gerente para este modelo é apenas responsável em apresentar os alarmes ao usuário do sistema. Após obter as informações, os gerentes determinam as ações a serem tomadas.

Os gerentes desenvolvidos para este modelo são os seguintes:

- **Gerente de Infraestrutura**: responsável pela infraestrutura das centrais locais (portas de acesso, umidade e temperatura dos locais onde se encontram os equipamentos). 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.
- **Gerente de Serviços**: responsável em gerenciar os serviços oferecidos aos usuários da rede.

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.

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. Correlacionador

Quando ocorre uma falha em algum dos equipamentos existentes numa rede, vários eventos podem ser gerados, alguns provenientes de equipamentos interligados ao que falhou. Devido ao grande número de eventos gerados pelos vários equipamentos, a localização exata do equipamento que gerou a falha torna-se uma tarefa difícil de ser realizada. Para diminuir o número de eventos, existe a técnica de correlação. 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 e métodos, já citados. A técnica adotada neste trabalho é a correlação por codificação, apresentada em [26]. 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.

No modelo, o correlacionador é responsável pela correlação dos eventos. O correlacionador possui um módulo responsável em receber e correlacionar os eventos, o *Main Correlator Module* (MCM). 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 15.

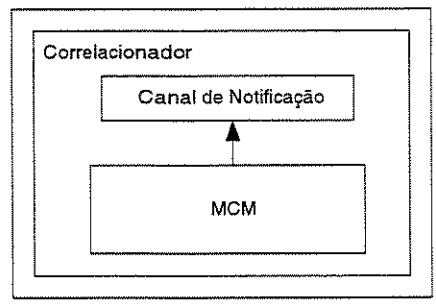


Figura 15: Correlacionador.

A utilização de um canal de notificação no correlacionador permite que mais de um gerente que tenha interesse em receber eventos correlacionados possa se conectar ao correlacionador. Outra vantagem é permitir que várias técnicas de correlação possam ser utilizadas, sem a necessidade de grandes alterações na forma como os eventos são transmitidos aos gerentes. A Figura 16 apresenta o correlacionador ligado aos gerentes e recebendo eventos.

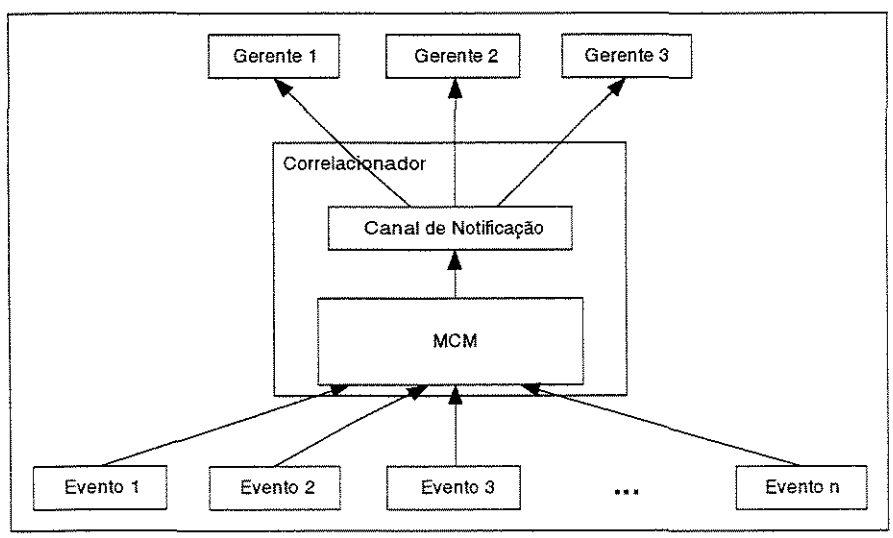


Figura 16: Correlacionador e Gerentes.

3.5. 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 caminhos nesta rede, como diamantes e ciclos. Um diamante é formado quando existem dois caminhos diferentes entre dois nós

da rede. Já um ciclo é um caminho onde um nó é percorrido duas vezes. Na Figura 17 é mostrado um exemplo de um diamante e um de um ciclo.

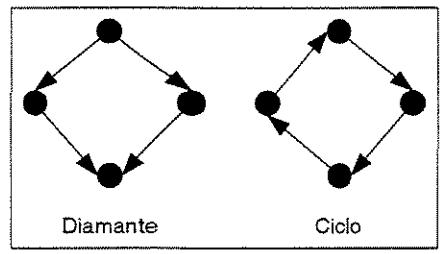


Figura 17: Exemplos de Diamante e Ciclo.

Para o modelo, este 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*.

3.6. Network Status

Para construir a rede utilizada na simulação 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. Além disso, o correlacionador as utiliza para determinar a ocorrência de falhas e os elementos da rede para simularem falhas.

O *Network Status* é um objeto servidor que mantém informações como: o número de centrais, o estado de cada equipamento de cada central e as conexões entre as centrais, que são utilizadas para a construção da rede .

Assim, o modelo final é apresentado na Figura 18, onde pode ser visto como todos os componentes são interligados. O componente log está descrito na Seção 3.8.

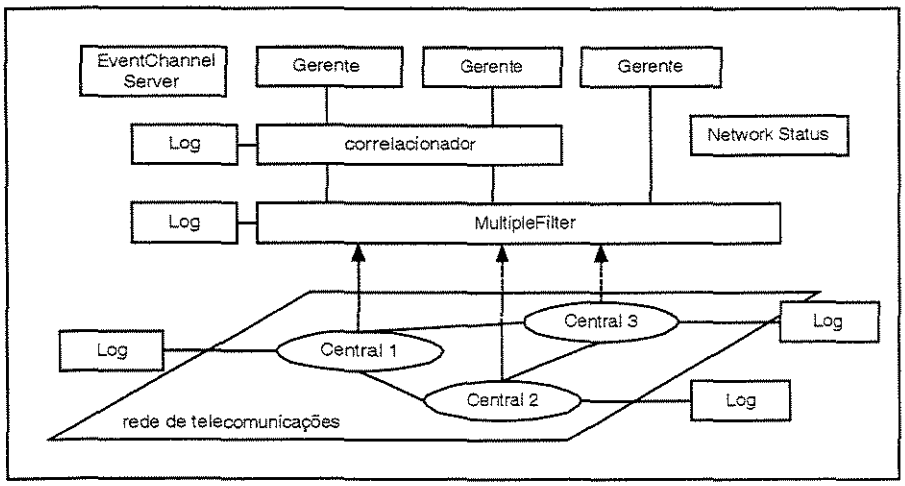


Figura 18: Modelo Proposto.

3.7. Eventos

Os alarmes são propagados no modelo através dos eventos estruturados. A construção dos eventos segue a especificação ITU-T X.733 [2], onde são determinados os campos e valores utilizados pelo modelo.

Os alarmes utilizados são apresentados na Tabela 2.

Tabela 2: Tipos de Alarmes.

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

Os atributos escolhidos para a utilização são mostrados na Tabela 3.

Tabela 3: Atributos do Alarme.

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

Estes atributos foram escolhidos pelo fato de serem os mais relevantes para o modelo, e por serem os utilizados em algumas ferramentas de gerenciamento de redes [8].

Para o modelo foi incluído mais um atributo: local, que indica em qual estação predial se encontra o equipamento que originou o evento. O atributo local é utilizado pelo correlacionador para dividir os eventos segundo o local de origem. 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.

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 pacotes de atributos condicionais, entre eles, pacote condicional de log finito, pacote condicional de escalonamento e pacote condicional de alarme. A Tabela 4 apresenta os atributos do pacote obrigatório e a Tabela 5, o pacote condicional de log finito.

Tabela 4: Pacotes de Atributos Obrigatórios de Log.

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 para 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 disponibilidade	Define a disponibilidade do log. Pode indicar uma condição de log cheio, impedindo o armazenamento de novos registros.

Tabela 5: Pacote Condicional de Log Finito.

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

Como o interesse deste trabalho é mostrar a viabilidade de incluir um sistema de log ao modelo, 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.

O modelo conta com dois tipos de log: um conectado ao canal da Central ou do *MultipleFilter*, registrando todos os eventos que transitam pelo canal e o outro conectado ao canal do Correlacionador, registrando os eventos gerados pela correlação dos eventos recebidos.

Para obter as informações do log é necessário acessar as bases de dados referentes ao canal na qual o log se encontra conectado. Desta forma é possível relacionar as informações entre as bases de dados para descobrir o caminho que o evento seguiu.

3.9. Visão Detalhada do Modelo

Após a descrição dos elementos presentes no modelo, uma visão mais detalhada do modelo é apresentada na Figura 19. Nesta visão podem ser vistos os canais de notificação e os outros objetos descritos.

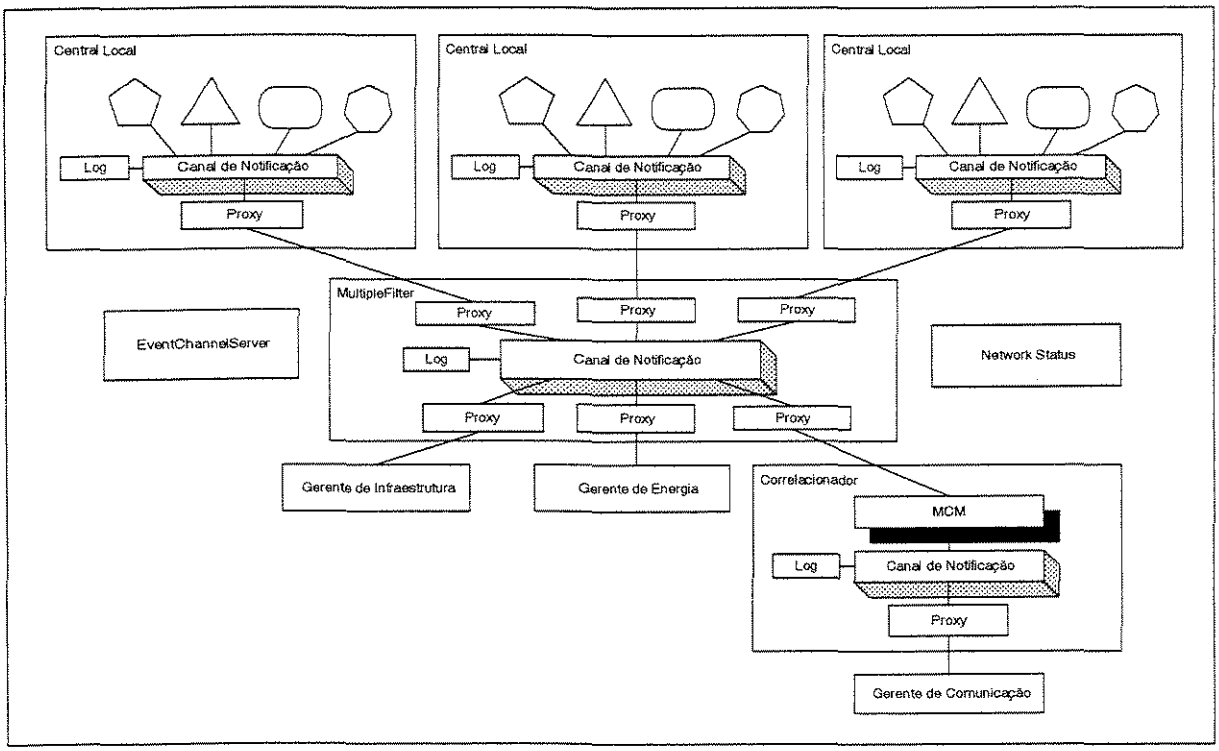


Figura 19: Visão Detalhada do Modelo.

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 relativas aos equipamentos existentes em cada central local.

Após iniciar a rede, cada central conecta seu canal ao canal do *MultipleFilter*. Para os testes, o estado de cada equipamento das centrais também é fornecido pelo *Network Status*, indicando se o equipamento está funcionando ou não. Para um caso real, o estado de cada equipamento é fornecido por ele mesmo. 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 MCM 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.

Para o gerente determinar se um conjunto de eventos gera um evento correlacionado ou se este conjunto deve ser descartado foi definido um intervalo de tempo. Este intervalo de tempo é iniciado a partir da chegada do primeiro evento de um código. Assim, se todos os eventos do conjunto estiverem presentes dentro do intervalo de tempo, o evento correlacionado será gerado. Caso o intervalo de tempo expire e o conjunto de eventos não esteja completo, este é descartado e um novo evento deste conjunto é aguardado para iniciar o intervalo de tempo.

Um gerente pode se conectar ao *MultipleFilter* ou ao correlacionador, dependendo se ele 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 passam por apenas dois caminhos. O primeiro pelo correlacionador, onde gerarão um novo evento se todos os eventos desejados estiverem presentes no correlacionador, caso contrário, os eventos são descartados e nenhum evento é gerado. O outro caminho é a conexão direta com o gerente, desta forma, os eventos vão do canal do MultipleFilter para o gerente, como mostrado na Figura 20.

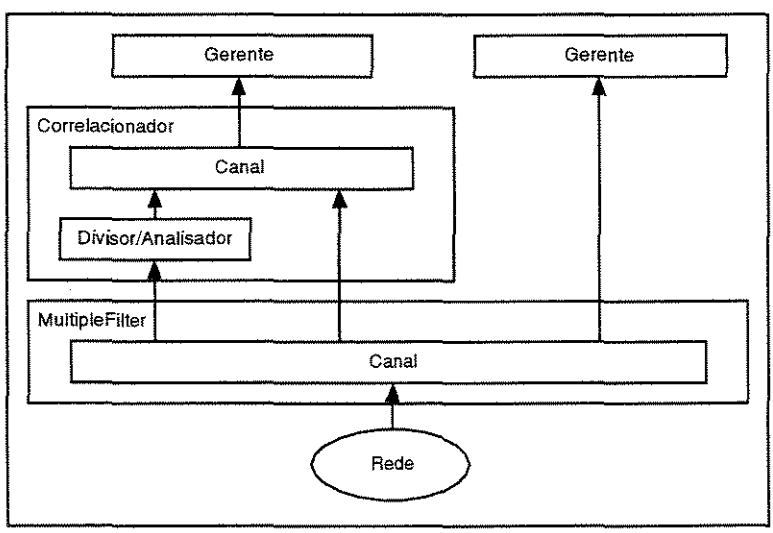


Figura 20: Gerente recebendo Eventos Correlacionados e Não Correlacionados.

3.10. Cenários

Nesta Seção são apresentados três cenários, onde o primeiro cenário mostra como um evento sem correlação trafega no sistema e o segundo, um evento com correlação. O terceiro cenário mostra como o evento é propagado através do Sistema de Log.

3.10.1. Cenário 1

O cenário 1 apresenta um exemplo de um evento que trafega dentro do modelo, sem a correlação de eventos. O evento utilizado é criado pelo gerador (*Power problem*), indicando a ocorrência de algum problema com o sistema de energia da central. O evento então é transmitido pelo canal da central até o canal do *MultipleFilter*, em seguida, o evento alcança os gerentes, como é apresentado na Figura 21.

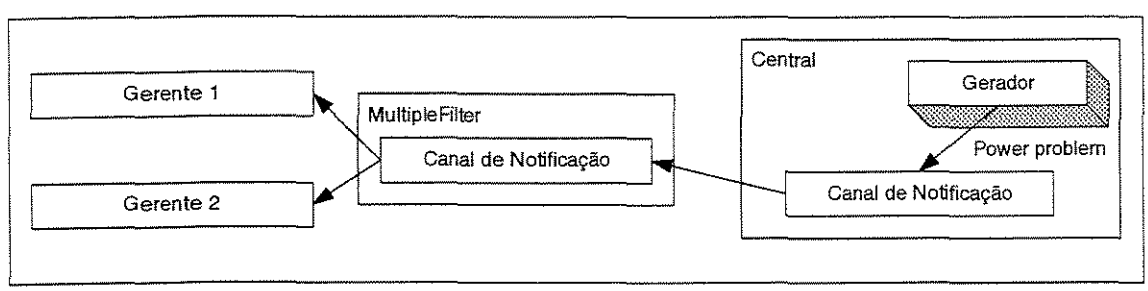


Figura 21: Cenário 1.

3.10.2. Cenário 2

Este cenário mostra como um evento correlacionado é gerado. Neste cenário, um equipamento de central telefônica está com falha e os equipamentos que dependem dele começam a gerar eventos, como mostrado na Figura 22 .

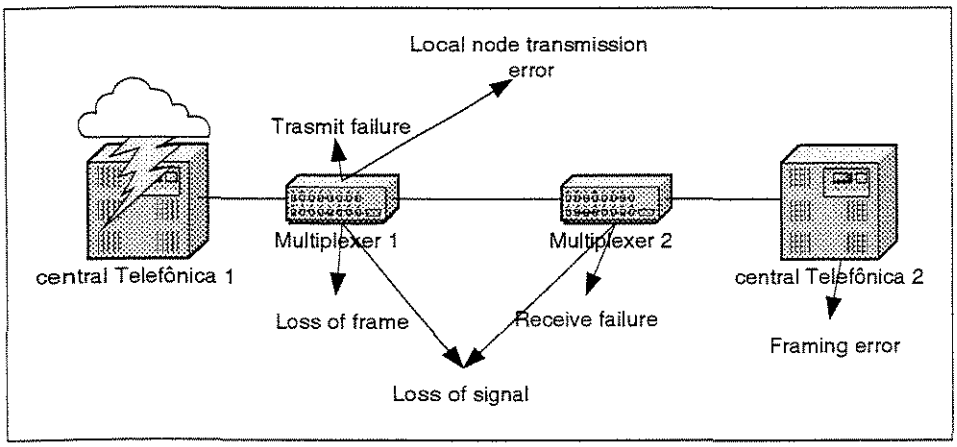


Figura 22: Central Telefônica com Falhas e Eventos Gerados.

O Correlacionador, então recebe os eventos, divide-os e checa o seu *codebook* para localizar alguma entrada que corresponda aos eventos recebidos. Após encontrar uma entrada que corresponde aos eventos recebidos, o Correlacionador envia o evento resultante para o gerente interessado em receber. A Figura 23 mostra a entrada e a saída do Correlacionador.

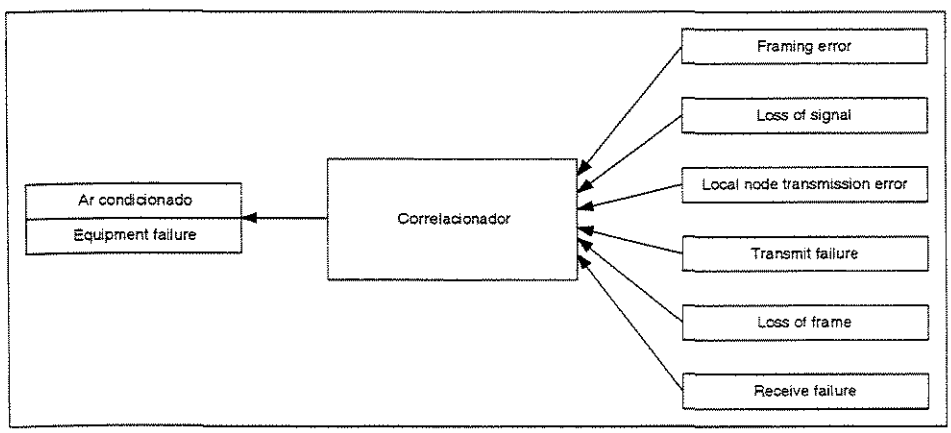


Figura 23: Resultado da Correlação.

3.10.3. Cenário 3

Este cenário mostra como um evento é armazenado no Sistema de Log. O evento ao ser enviado por um dos equipamentos vai para o canal da central que possui um log. Este log armazena o evento, mantendo como um dos eventos gerados pela central. Em seguida, o evento é retransmitido para o canal do *MultipleFilter*.

Como podem existir várias centrais ligadas ao *MultipleFilter*, o evento é novamente armazenado no log do *MultipleFilter*. A partir do canal do *MultipleFilter*, o evento tem dois caminhos que pode seguir: o primeiro, o evento é recebido pelos gerentes que possuem interesse nele, e o segundo o evento é enviado para o correlacionador. Neste segundo caso, o correlacionador pode utilizá-lo para gerar um novo evento. Este novo evento é transmitido para o canal do correlacionador e armazenado no log do correlacionador. Este cenário é mostrado na Figura 24.

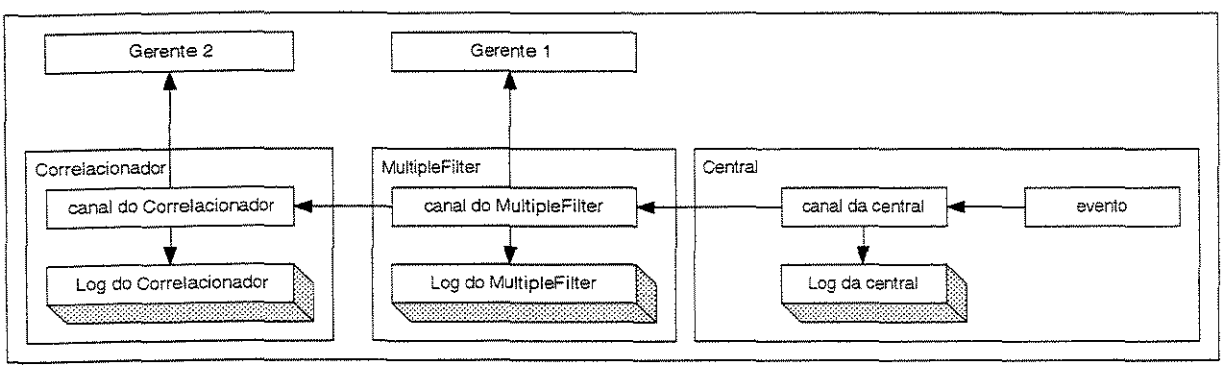


Figura 24: Cenário 3.

3.11. Trabalhos Relacionados

O trabalho desenvolvido em [8] 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 este trabalho apresenta 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.

Utilizando-se das facilidades oferecidas pela arquitetura CORBA, a empresa *Hewllet-Packard* desenvolveu a ferramenta *HP OpenView CORBA TMN Gateway* [7] que permite que gerentes CORBA se comuniquem com agentes CMIP. Por se tratar de uma ferramenta comercial, a empresa *Hewllet-Packard* desenvolveu apenas um adaptador de objetos, da mesma forma que o trabalho desenvolvido em [8].

O trabalho apresentado em [22] mostra que existe a viabilidade de se construir um sistema de gerenciamento para rede de telecomunicações utilizando a arquitetura CORBA com todos os componentes existentes na arquitetura TMN. Já este trabalho, mostra que nem todos os componentes da arquitetura TMN são necessários, sendo possível que a própria interface do objeto gerenciado seja um agente.

Capítulo 4

Implementação do Modelo Proposto

Para a implementação foi utilizado o ORB OrbixWeb 3.1c da IONA, JDK 1.2.1 da Sun e o pacote de serviços Openfusion versão 1.1.0 da Prismtech. Este pacote oferece os serviços de Nome, *Trading*, Eventos, Notificação, Ciclo de Vida e Propriedades. Quanto ao hardware, foi utilizada uma estação de trabalho SUN com o sistema operacional Solaris 2.7.

As interfaces dos produtores e consumidores com seus respectivos métodos são descritos na Seção 4.1. A Seção 4.2 apresenta uma breve descrição quanto à implementação de cada componente do modelo e a Seção 4.3 apresenta os testes realizados com o modelo.

4.1. Interfaces Utilizadas

As interfaces *StructuredPushConsumer* e *StructuredPullConsumer* são utilizadas para definir os consumidores de eventos estruturados do tipo *push* e do tipo *pull*, respectivamente. Estas interfaces possuem os métodos que os consumidores utilizarão para obter os eventos e se desconectarem. As interfaces em IDL são apresentadas na Figura 25 .

```
interface StructuredPushConsumer : NotifyPublish {

    void push_structured_event(in CosNotification::StructuredEvent notification)
      raises(CosEventComm::Disconnected);

    void disconnect_structured_push_consumer();

}; // StructuredPushConsumer

interface StructuredPullConsumer : NotifyPublish {

    void disconnect_structured_pull_consumer();

}; // StructuredPullConsumer
```

Figura 25: Interfaces StructuredPushConsumer e StructuredPullConsumer.

Como pode ser observado, a interface *StructuredPushConsumer* possui o método *push_structured_event* para receber os eventos, enquanto que a interface *StructuredPullConsumer* não possui um método para obter os eventos. Isto se deve à forma

como os eventos são obtidos, quando utilizando a técnica *push*, o *proxy* ligado ao consumidor invoca o método *push_structured_event* do próprio consumidor. Já para a técnica *pull*, o consumidor tenta obter o seu evento utilizando os métodos do seu *proxy*.

Ambas as interfaces herdam a interface *NotifyPublish* que permite através da operação *offer_change* que os consumidores informem aos produtores quais os eventos que devem ser produzidos e os que deixam de ser produzidos. A interface de *NotifyPublish* é apresentada na Figura 26.

```
Interface NotifyPublish {
    void offer_change (in CosNotification::EventTypeSeq added,
      in CosNotification::EventTypeSeq removed )
      raises ( InvalidEventType );
}; // NotifyPublish
```

Figura 26: Interface NotifyPublish.

Da mesma forma que os consumidores, os produtores utilizam interfaces para a implementação. As duas interfaces são mostradas na Figura 27. A interface *StructuredPullSupplier* é utilizada pelos produtores de eventos estruturados que transmitem utilizando o método *pull* e a interface *StructuredPushSupplier* para o método *push*.

```
Interface StructuredPullSupplier : NotifySubscribe {

    CosNotification::StructuredEvent pull_structured_event()
      Raises(CosEventComm::Disconnected);

    CosNotification::StructuredEvent try_pull_structured_event(out boolean has_event)
      Raises(CosEventComm::Disconnected);

    void disconnect_structured_pull_supplier();
}; // StructuredPullSupplier

interface StructuredPushSupplier : NotifySubscribe {

    void disconnect_structured_push_supplier();

}; // StructuredPushSupplier
```

Figura 27: Interfaces StructuredPullSupplier e StructuredPushSupplier.

Para produzir os eventos, os produtores do tipo *pull* utilizam os métodos *pull_structured_event* e *try_pull_structured_event* para enviar os eventos. Estes métodos são invocados pelo *proxy* a que está conectado. Já os produtores do tipo *push*, o produtor simplesmente envia o evento invocando o método *push_structured_event* do seu *proxy*.

Ambas as interfaces produtoras herdam a interface *NotifySubscribe* que possui o método *subscription_change* que permite aos consumidores adicionar ou remover eventos de seu interesse. A interface IDL *NotifySubscribe* é apresentada na Figura 28.

```
interface NotifySubscribe {

    void subscription_change(in CosNotification::EventTypeSeq added,
      in CosNotification::EventTypeSeq removed )
      raises ( InvalidEventType );

}; // NotifySubscribe
```

Figura 28: Interface NotifySubscribe.

4.2. Implementação dos Componentes do Modelo Proposto

A seguir, será apresentada uma breve descrição dos componentes do modelo proposto apresentado no Capítulo 3.

4.2.1. EventChannelServer

É o objeto responsável em receber as chamadas para a criação de canais e de conexões entre os canais, assim como a remoção dos mesmos. Este objeto mantém um mapa com as conexões entre os canais. Este mapa é implementado através de uma matriz, onde as linhas e colunas representam os identificadores dos canais de notificação e a intersecção entre a linha e a coluna retorna um valor booleano indicando a ocorrência de uma conexão ou não. Desta forma apenas uma conexão é permitida de um canal para outro.

Quanto aos pedidos de criação de canais, o *EventChannelServer* retorna o identificador do canal e o próprio canal, utilizando uma estrutura com estes dois campos, o *Cid*, como mostrado na Figura 29 . Desta forma, a aplicação que pede a criação de um canal recebe o canal e seu respectivo identificador.

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

Figura 29 : Estrutura Cid.

No caso da criação de uma conexão, o *EventChannelServer* retorna apenas um identificador indicando a conexão, mas armazena outras informações importantes, na classe *connection*, como mostrado na Figura 30 . Para criar uma conexão é necessário definir o

canal produtor, o canal consumidor e o tipo de conexão que será feita (*push* ou *pull*). O *EventChannelServer* mantém duas listas, uma armazenando os canais e seus identificadores e a outra, as conexões.

```
struct connection{
        long id;
        ProxyType type;
        long supplier;
        long consumer;
        CosNotifyChannelAdmin::StructuredProxyPushConsumer proxypushConsumer;
        CosNotifyChannelAdmin::StructuredProxyPushSupplier proxypushSupplier;
        CosNotifyChannelAdmin::StructuredProxyPullConsumer proxypullConsumer;
        CosNotifyChannelAdmin::StructuredProxyPullSupplier proxypullSupplier;
};
```

Figura 30 : Estrutura connection para armazenar informações sobre a conexão entre dois canais.

Além de criar canais e conexões, o *EventChannelServer* possui métodos para retornar os canais produtores e consumidores das centrais locais referentes ao *MultipleFilter*, todas as conexões existentes e o identificador do canal do *MultipleFilter*. A IDL completa do *EventChannelServer* é apresentada na Figura 31.

```
Module ChannelServer{
typedef sequence<long> sequencelong;
typedef sequence<CosNotification::Property> seqProperty;
typedef sequence<CosNotification::EventType> seqEventType;

enum ProxyType {
    PUSH_ANY, PULL_ANY, PUSH_STRUCTURED, PULL_STRUCTURED, PUSH_SEQUENCE,
    PULL_SEQUENCE, PUSH_TYPED, PULL_TYPED
  };

struct connection{
        long id;
        ProxyType type;
        long supplier;
        long consumer;
        CosNotifyChannelAdmin::StructuredProxyPushConsumer proxypushConsumer;
        CosNotifyChannelAdmin::StructuredProxyPushSupplier proxypushSupplier;
        CosNotifyChannelAdmin::StructuredProxyPullConsumer proxypullConsumer;
        CosNotifyChannelAdmin::StructuredProxyPullSupplier proxypullSupplier;
  };

typedef sequence<connection> seqconnection;

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

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 31 : IDL do EventChannelServer.

4.2.2. MultipleFilter

O *MultipleFilter* é o canal principal, por onde os outros canais se interligam. O seu canal é criado através do *EventChannelServer*. O canal criado por este objeto é a ponte entre os canais existentes e os gerentes. Sua principal função é interligar os gerentes e os canais das centrais. A sua interface IDL é mostrada na Figura 32.

```
Module MultipleFilter{
        typedef sequence<CosEventChannelAdmin::EventChannel> seqChannel;
        typedef sequence<long> seqlong;
        typedef sequence<CosNotification::EventType> seqEventType;

        interface multFilter{
                CosNotifyChannelAdmin::EventChannel return_channel_filter();
                void set_all_filters(in seqEventType types, in string constraint);
                seqlong return_all_connected_supplier_channels();
                void connect_channel(in long who, in seqEventType intypes) raises
(CosNotifyChannelAdmin::ChannelNotFound);
                boolean disconnect_channel(in long who) raises
(CosNotifyChannelAdmin::ChannelNotFound);
        };
};
```

Figura 32: Interface de MultipleFilter.

4.2.3. Central

A central é formada por *threads* que simulam o funcionamento dos agentes dos equipamentos (ar condicionado, desumidificador, gerador, multiplexador e central telefônica) e um canal de notificação que se liga ao canal do *MultipleFilter*. O estado de cada equipamento é mantido pelo *Network Status*. Cada *thread* é um produtor de eventos do tipo *push* (*StructuredPushSupplier*) que obtém a informação de seu estado do *Network Status*, gerando eventos ou não. Cada agente se interliga ao canal de notificação, por onde os eventos serão enviados para os gerentes.

Outras vezes, o evento pode ser gerado em decorrência de uma falha em outro equipamento. Para isso o agente pode obter o estado do equipamento adjacente ao seu, gerando os eventos causados pela falha deste equipamento.

4.2.4. Correlacionador

O correlacionador é formado por um canal por onde os gerentes podem se conectar a ele, 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 receptor de eventos, um consumidor do tipo *pull*, que se conecta ao canal do *MultipleFilter* para receber todos os eventos produzidos pelas centrais, dividindo os eventos por centrais. Esta divisão é baseada na topologia da rede, de tal forma que uma central receba os eventos gerados por ela mesma e pelas centrais adjacentes.
- O analisador é a parte responsável em analisar os eventos divididos, correlacionando-os e gerando um novo evento representando a verdadeira falha. O analisador é um produtor de eventos do tipo *push*.

Apesar do *correlator* ser descrito como dois módulos, sua implementação utiliza apenas de uma classe. A interface do correlacionador herda as características das duas interfaces utilizadas: *StructuredPullConsumer* e *StructuredPushConsumer*. A interface é apresentada na Figura 33.

```
Module Correlation{
typedef sequence<CosNotification::EventType> seqEventType;

        interface correlatorServer{
                CosNotifyChannelAdmin::EventChannel return_channel();
                long return_id();
                void create_correlator(in CosNotification::EventType intypes);
                boolean destroy_correlator(in long id);
        };

        interface
correlator:CosNotifyComm::StructuredPullConsumer,CosNotifyComm::StructuredPushSupplier{
        };
};
```

Figura 33 : Interface do Correlacionador.

A técnica de correlação adotada para este trabalho é a correlação por codificação. O *codebook* é uma lista de eventos resultantes da correlação e cada evento possui seu código, mantido por um conjunto de bits, onde cada posição indica um evento que deve ser recebido. Um exemplo de *codebook* é apresentado na Figura 34.

	Equipamento			Ambiente				Comunicação						
	Problema no Multiplexador	**Equipamento com mal funcionamento**	**Problema de energia**	**Problema no sistema de H/V/C**	**Temperatura inaceitável**	**Umidade inaceitável**	**Porta Aberta**	**Perda de Sinal**	**Erro local de Transmissão**	**Perda de *frame***	**Falha de Transmissão**	**Falha no Transmissor**	**Falha de Recepção**	**Falha no Receptor**
Problema no Multiplexador, multiplex	1	1	0	0	0	0	0	1	1	1	1	1	1	1
Equipamento com mal funcionamento, Central	0	1	0	0	0	0	0	1	0	1	1	1	1	1
Erro Local de Transmissão, Tronco	0	0	0	0	0	0	0	1	1	1	1	1	1	1
Problema no sistema de H/V/C, Ar condicionado	0	1	0	1	1	0	0	0	0	0	0	0	0	0
Problema no sistema de H/V/C, Umidificador	0	1	0	1	0	1	0	0	0	0	0	0	0	0
Porta Aberta, Porta	0	0	0	0	1	1	1	0	0	0	0	0	0	0
Problema de energia, Gerador	0	1	1	0	0	0	0	0	0	0	0	0	0	0

Figura 34 : Exemplo de um *Codebook*.

Para o exemplo apresentado na Figura 34, o problema "Problema no Multiplexador" tem o código 11000001111111 e será a origem dos seguintes eventos (sintomas): Problema no Multiplexador, Equipamento com Mal Funcionamento, Perda de sinal, Erro Local de Transmissão, Perda de *frame*, Falha de Transmissão, Falha no Transmissor, Falha de Recepção e Falha no Receptor.

O funcionamento do correlacionador é o seguinte, após receber o evento, o receptor determina para quais centrais o evento deve ser enviado. O evento então é armazenado na lista de eventos da central. A seguir, o MCM verifica se todos os eventos de cada central geram um código igual a algum do *codebook*. Se sim, um evento é gerado e enviado para o canal do correlacionador, e os eventos que originaram o evento correlacionado são removidos.

Caso a tabela de códigos alcance um tamanho muito grande dificultando o processamento, é possível eliminar os eventos menos significativos da tabela. Os eventos

menos significativos são escolhidos através de uma seleção, atendendo um dos seguintes critérios:

- evento gerado representa um sintoma comum a várias causas, de forma que sua ocorrência não restringe o número de possíveis causas.
- evento gerado representa um sintoma de poucas causas, quase sempre sem peso para representar a causa.

Utilizando estes itens é possível diminuir o tamanho da tabela de códigos, garantindo que apenas os eventos que realmente possuem importância para a correlação se encontrem na tabela.

4.2.5. Network Status

Network Status é um servidor CORBA que mantém as informações da rede, tais como o número de centrais, o estado de cada equipamento de cada central, o número de centrais e a topologia da rede. Pode obter também quais as centrais adjacentes a uma central e aumentar o tamanho da rede. É implementado como um simples servidor de informações. É utilizado para fornecer as informações para os outros programas e gerenciá-las. A interface do *Network Status* é apresentada na Figura 35.

```
module Net_Status{
        typedef sequence <long> seqLong;
        interface net_statusServer{
                boolean connection(in long de,in long para);
                void add_ct(); //adiciona uma central a rede
                long return_size();
                void set_size(in long size);
                seqLong get_all_neighbors(in long ct);
                boolean read_status(in long ct, in long eqp);
                void write_status(in long ct, in long eqp);
        };
};
```

Figura 35 : Interface IDL de Network Status.

4.2.6. Sistema de Log

O log para o modelo é um consumidor de eventos do tipo *push* que se conecta ao canal e captura todos os eventos, armazenando-os. Caso seja necessário, pode se anexar ao log, um filtro para selecionar apenas eventos de interesse ao sistema. Da mesma forma, vários logs podem ser colocados em um mesmo canal. Existem três tipos de log:

- Log de Central: log conectado ao canal da central. Armazena todos os eventos gerados pelos equipamentos existentes na central.

- Log do *MultipleFilter*: log conectado ao canal do *MultipleFilter*. Armazena todos os eventos que trafegam no canal do *MultipleFilter*.
- Log do Correlacionador: log conectado ao canal do Correlacionador. Armazena os eventos gerados pela correlação de outros eventos.

Cada evento recebido gera um registro no log. Como a principal função do log é armazenar os registros em um sistema persistente, os eventos são armazenados numa base de dados, onde cada log ligado a um canal gera uma tabela onde os eventos serão armazenados, facilitando a recuperação e a consulta. A base de dados escolhida foi o MySQL para Solaris. Quanto ao driver JDBC utilizado para a conexão com a base de dados, foi escolhido o driver MM.MySQL 2.0.1. A Figura 36 apresenta dois exemplos de registros do log.

```
RegisterID: 0
LogTime: 979647156025
Domain: Telecommunication
Type: Enviromental
Name: Heating/Ventilation/Cooling
system problem
ManagedObjectClass: refrigeracao
ManagedObjectInstance: refrigeracao1
EventTime: 979647154828
ProbableCause:
PerceivedServerity:
Local: Central number 1
AddittionalText:
```

```
RegisterID: 53
LogTime: 979647175157
Domain: Telecommunication
Type: Equipment
Name: Equipment malfunction
ManagedObjectClass: ar condicionado
ManagedObjectInstance: ar
condicionado1
EventTime: 979647175150
ProbableCause:
PerceivedServerity:
Local: Central number 1
AddittionalText:
```

Figura 36: Exemplos de registros do log.

4.2.7. Gerentes

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

No modelo proposto existem quatro gerentes: de infraestrutura, de energia, de comunicação e de serviços. Para cada gerente é definido um filtro para que o gerente receba apenas os eventos de seu interesse. Desta forma, os gerentes têm filtros como mostrado na Tabela 6.

Tabela 6: Gerentes e Filtros.

Gerente	Filtros
Infraestrutura	$domain_name == 'Telecommunication' and $type_name == 'Environmental'
Energia	$domain_name == 'Telecommunication' and $type_name == 'equipment' and $event_name == 'Power Problem'
Comunicação	$domain_name == 'Telecommunication' and $type_name == 'Communications' $.filterable_data(managedObjectClass) == 'Multiplexer' $.filterable_data(managedObjectClass) == 'CT'
Serviços	$domain_name == 'Telecommunication' and $type_name == 'Quality of Service'

Os gerentes não precisam estar no mesmo centro de operações, sendo possível manter vários gerentes em diferentes centros, garantindo assim que sempre haverá um gerente supervisionando em caso de falhas.

4.2.8. Eventos

Todos os alarmes ocorridos nos equipamentos são transmitidos pelos eventos estruturados do Serviço de Notificação. O evento estruturado, os campos utilizados, como apresentados na Seção 3.7, e seus respectivos tipos são apresentados na Figura 37.

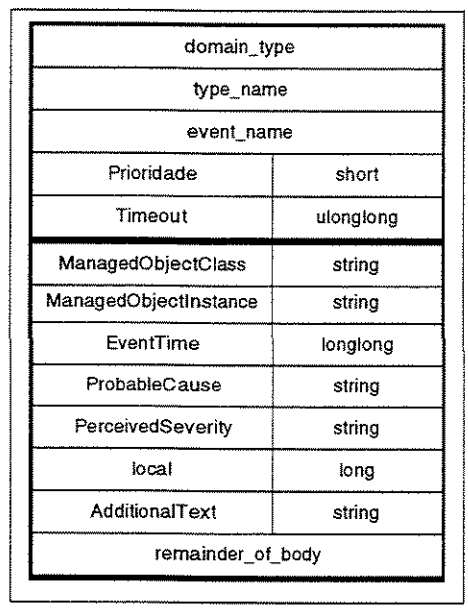


Figura 37: Estrutura do Evento Utilizado.

Para a produção dos eventos, os produtores utilizam o método *factory_events* da classe *util* para a criação do evento. O método é apresentado na Figura 38.

```
synchronized StructuredEvent factory_events(fields flds)
{
        org.omg.CORBA.Any data;
        Date now = new Date();
        StructuredEvent event = new StructuredEvent();

        event.header = new EventHeader();
        event.header.fixed_header = new FixedEventHeader();
        EventType type = new EventType(flds.domain, flds.type);
        event.header.fixed_header.event_type = type;
        event.header.fixed_header.event_name = flds.name;

        short prio = (short) 50;
        event.header.variable_header = new Property[2];
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_short(prio);
        event.header.variable_header[0] = new Property(Priority.value, data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_ulonglong(10*1000*1000*60*5); // 5 minutes
        event.header.variable_header[1] = new Property(Timeout.value, data);

        event.filterable_data = new Property[7];
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_string(flds.managedObject);
        event.filterable_data[0] = new Property("managedObjectClass", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_string(flds.managedInstance);
        event.filterable_data[1] = new Property("managedObjectInstance", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_longlong(now.getTime());
        event.filterable_data[2] = new Property("eventTime", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_string(flds.cause);
        event.filterable_data[3] = new Property("probableCause", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_string(flds.severity);
        event.filterable_data[4] = new Property("perceivedSeverity", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_long(flds.local);
        event.filterable_data[5] = new Property("local", data);
        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_string(flds.addtext);
        event.filterable_data[6] = new Property("additionalText", data);
        event.remainder_of_body = org.omg.CORBA.ORB.init().create_any();
        event.remainder_of_body.insert_string("remainder of body");
        return event;
}
```

Figura 38 : Método factory_events.

Como pode ser visto, o método tem como parâmetro de entrada um objeto da classe *fields*, composto pelos campos utilizados para preencher a estrutura do evento estruturado.

Quanto à transmissão dos eventos para o canal, isso é feito através de uma invocação ao método *push_structured_event* do *ConsumerProxy* a qual o produtor está conectado. A Figura 39 mostra um exemplo desta invocação.

```
...
StructuredProxyPushConsumer proxy;
...
        try{
                proxy.push_structured_event(factory_events(flds);
        }catch( org.omg.CosEventComm.Disconnected ex)
        {
        ...
```

Figura 39: Invocação do Método push_structured_event.

4.3. 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 utilizados dois modelos. O primeiro é o Modelo Proposto (Figura 19) com doze canais conectados, onde dez canais se conectam com os produtores, e o outro é o Modelo Simplificado, onde todos os equipamentos estão ligados ao canal de notificação do *MultipleFilter*, sem a utilização de canais locais nas centrais. Os esquemas destes modelos são apresentados na Figura 40.

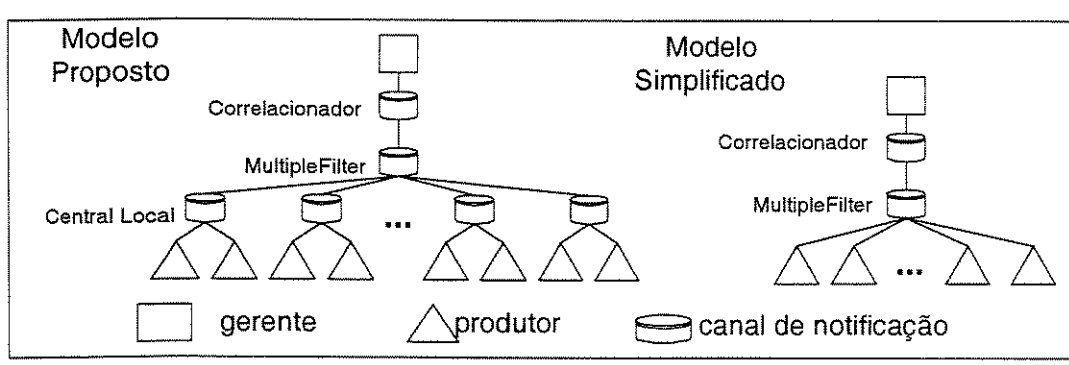


Figura 40: Representação dos Modelos para os Testes.

O primeiro teste consiste em analisar o comportamento com relação ao aumento de produtores de eventos no sistema. Para isso foram utilizados um gerente e vários conjuntos de produtores (de 30 a 150 produtores). A Figura 41 apresenta alguns resultados quanto ao tempo necessário para o gerente receber os eventos.

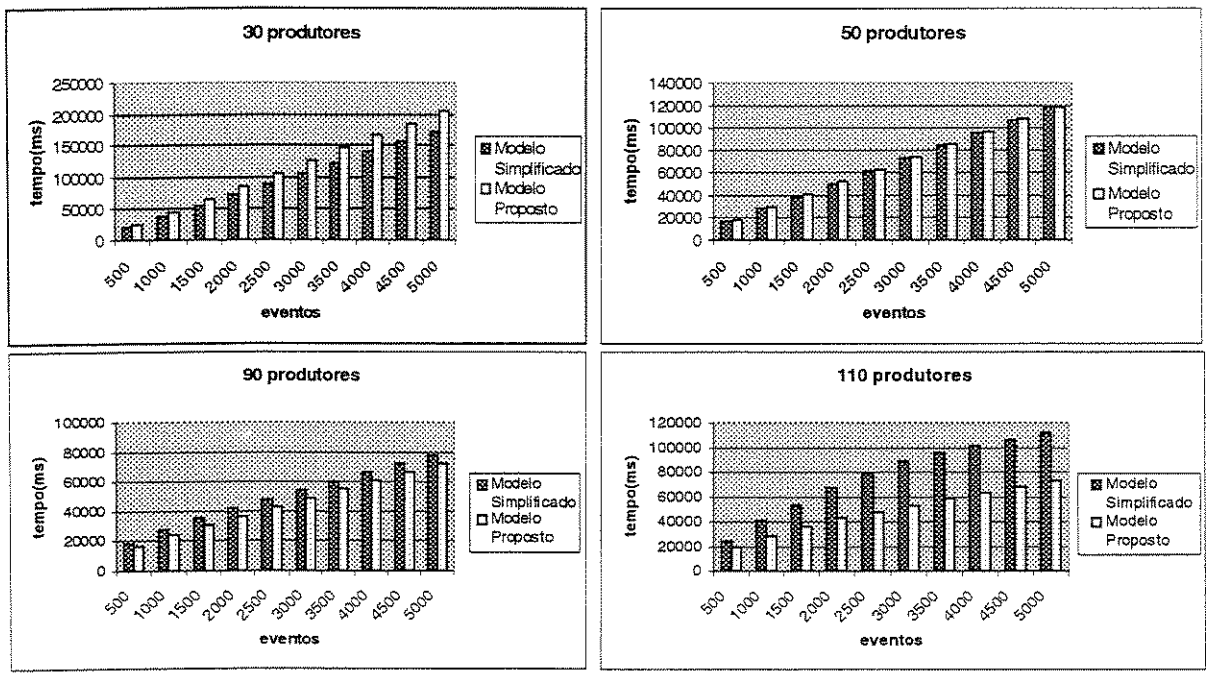


Figura 41: Tempo Total para 30, 50, 90 e 110 produtores.

Utilizando poucos produtores (de 30 a 40 produtores), o desempenho do Modelo Proposto ficou um pouco abaixo do Modelo Simplificado. O tempo total chegou a ser 20% a mais do que o Modelo Simplificado. Este fato se deve à sub-utilização dos canais de eventos. O melhor desempenho foi obtido com o intervalo entre 50 a 110 produtores, onde o tempo total foi de até 50% abaixo do tempo total do Modelo Simplificado. Este melhor desempenho foi causado pelo balanceamento entre o tempo de processamento dos eventos pelo gerente e pelo tempo de tráfego. Acima deste intervalo, o desempenho não difere muito do desempenho do Modelo Simplificado, onde o tempo total para o Modelo Proposto fica em torno de 10% melhor/pior que o Modelo Simplificado, como apresentado na Figura 42. Neste caso, existe uma quantidade maior de eventos trafegando, ocasionando um intervalo de tempo maior entre o envio e a recepção.

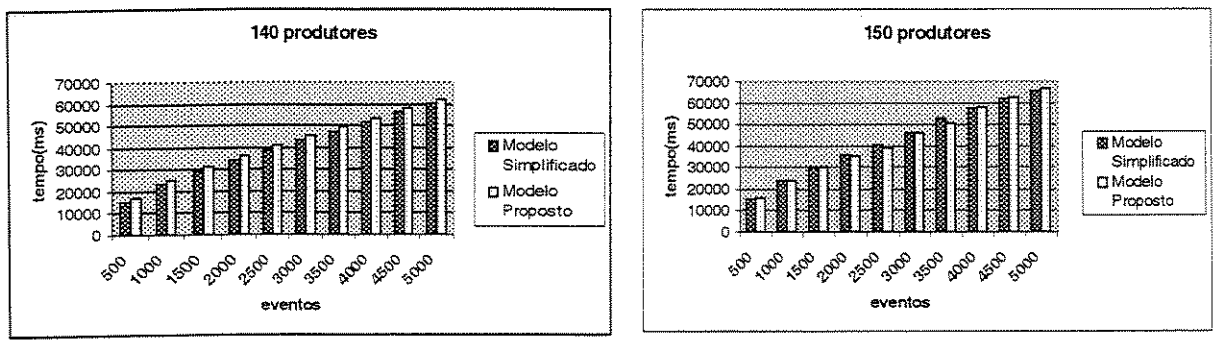


Figura 42: Tempo Total para 140 e 150 produtores.

A Figura 43 apresenta o tempo necessário para que o modelo proposto 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 44 apresenta o resultado com relação ao tempo de tráfego para o mesmo teste.

A causa das duas observações é o aumento do número de eventos. Por um lado, existe uma quantidade maior de eventos em um tempo menor para que o gerente trate, e por outro lado os mesmos eventos ficam aguardando um tempo maior para serem processados, pois a quantidade de eventos que chega ao gerente é maior que a quantidade que ele pode tratar.

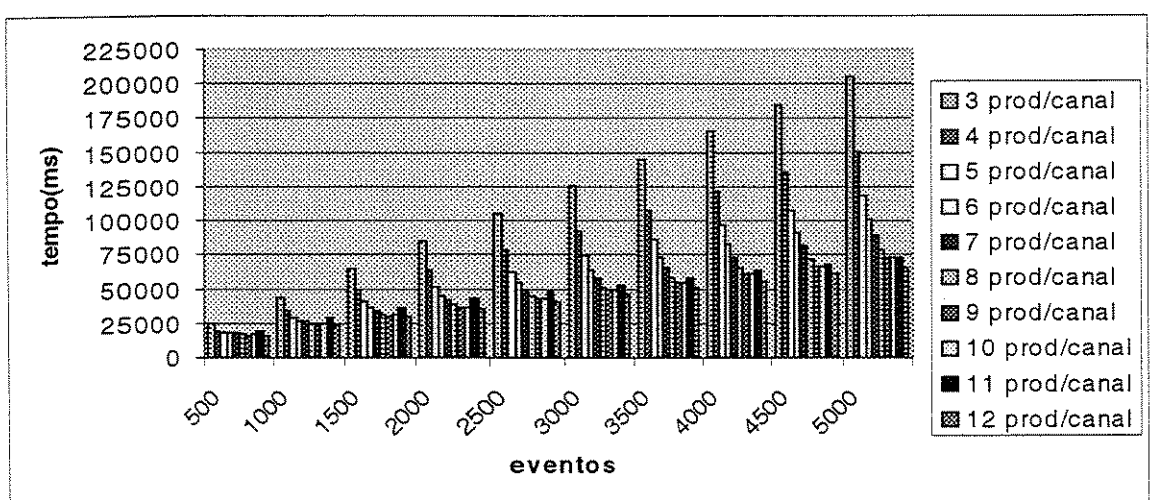


Figura 43: Tempo Total para o gerente receber os eventos, variando o número de produtores por canal.

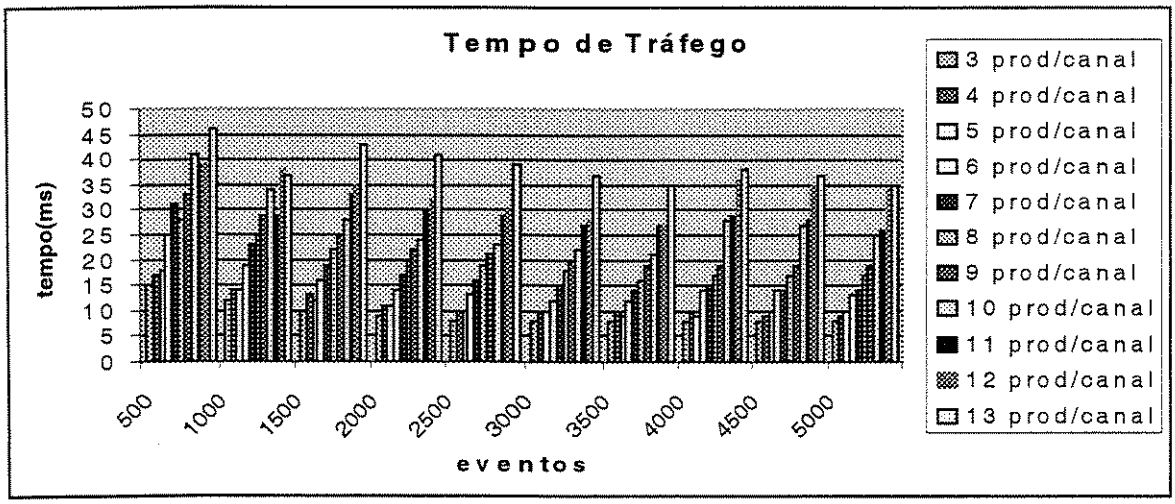


Figura 44: Tempo de Tráfego.

O segundo teste analisa o comportamento do Modelo Proposto com o aumento no número de canais ligados aos produtores, mantendo fixo o número total de produtores. A Tabela 7 apresenta os valores utilizados para os testes.

Tabela 7: Casos utilizados no Segundo Teste.

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

Os resultados obtidos para o segundo teste estão nas Figuras 45 e 46. 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.

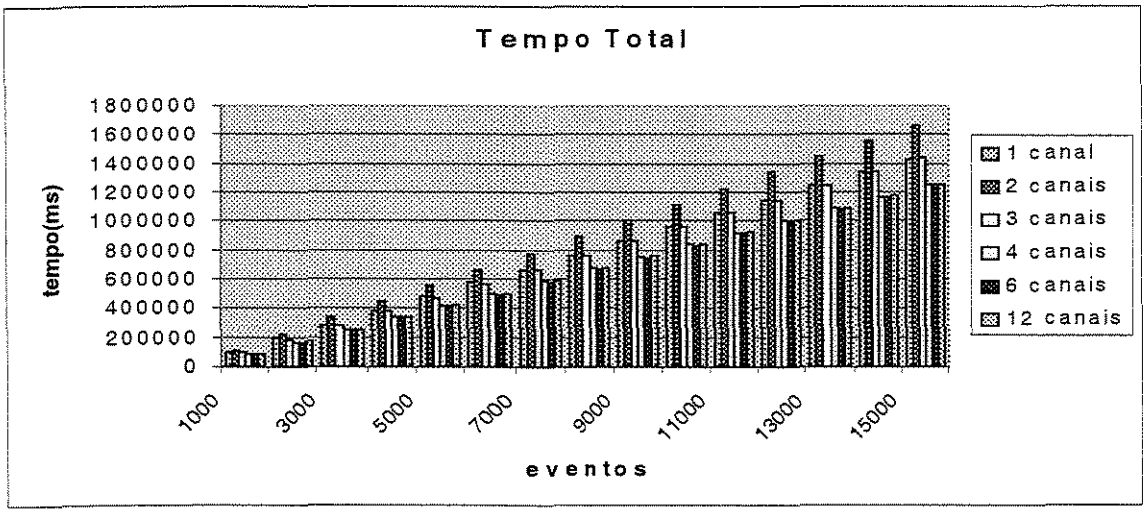


Figura 45: Tempo Total para o Segundo Teste.

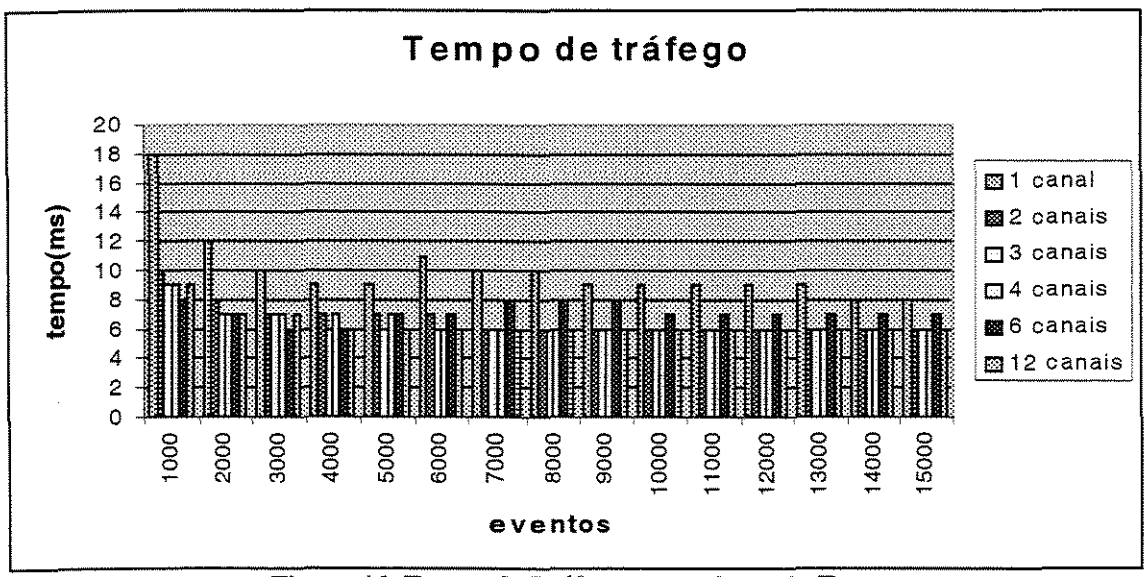


Figura 46: Tempo de Tráfego para o Segundo Teste.

A partir destes testes foi possível obter os seguintes resultados:

- O Modelo Proposto 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.

Capítulo 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 arquitetura TMN foi desenvolvida para permitir a integração dos sistemas de gerenciamento de redes de telecomunicações. Esta arquitetura apresenta certas dificuldades que impedem a sua utilização e a integração entre as aplicações de gerenciamento.

Já a arquitetura CORBA foi desenvolvida para permitir a interoperabilidade entre as aplicações de diferentes sistemas operacionais e implementados em linguagens diferentes. Esta arquitetura possui suporte para diversos tipos de aplicações, incluindo as de telecomunicações.

A integração das duas tecnologias fornece uma base sólida para o desenvolvimento de aplicações para o gerenciamento de redes de telecomunicações. Desta forma, o gerenciamento de redes de telecomunicações pode se beneficiar dos recursos e facilidades oferecidos pela arquitetura CORBA.

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 modelo desenvolvido apresenta um exemplo da possibilidade de integração dos módulos TMN com os módulos do Serviço de Notificação CORBA, para realizar a tarefa de gerenciamento de redes de telecomunicações.

A implementação apresenta as vantagens da utilização da arquitetura CORBA para o desenvolvimento dos módulos do modelo, assim como 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. A versão do pacote Openfusion utilizado ofereceu todos os serviços necessários para o desenvolvimento do modelo, uma vez que estes serviços não fazem parte da implementação básica da arquitetura CORBA.

Os testes e os resultados obtidos mostram que o Modelo Proposto é escalável com relação ao número de canais e conexões entre os canais. Os testes mostram também a existência de um intervalo no número de produtores onde o desempenho do modelo proposto é melhor. Pode-se concluir através dos testes que o modelo desenvolvido é escalável, permitindo a inclusão de novos canais sem a necessidade de alterações na estrutura do modelo. A inclusão de novos canais permite também o aumento do número de centrais e equipamentos dentro do modelo.

Baseado no intervalo no número de produtores onde o desempenho do modelo proposto é melhor, um trabalho futuro proposto é o desenvolvimento de uma rede de canais substituindo o canal do *MultipleFilter*. Esta rede seria montada em tempo de execução, atendendo aos requisitos de qualidade de serviço e do desempenho do sistema, tentando sempre manter um número ótimo de produtores de eventos por canal. A rede de canais também permitiria a abrangência de uma região maior ou de várias microrregiões. Atribuindo uma certa "inteligência" ao sistema, os canais das centrais seriam capazes de detectar um caminho melhor ou com menor tráfego. Outro trabalho que pode ser desenvolvido é a integração deste sistema com os sistemas já existentes, que utilizam os protocolos SNMP e CMIP.

Para o caso de redes móveis, as mudanças a serem feitas se relacionam ao tipo de equipamento utilizado pelas estações de rádio-base. O modelo, no geral, não é alterado necessitando apenas conectar os equipamentos das estações ao respectivo canal de notificação existente para cada estação.

Referências Bibliográficas

[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] **Glass, S.**. "Telecommunications Systems: An Introductory Guide". Fevereiro, 1997. http://www.gtlaw.com.au/pubs/telcosysintroguide.html

[7] **Hewlett Packard Company**. "HP OpenView Communications: CORBA TMN Gateway". Dezembro, 1999. http://www.hp.com.

[8] **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.

[9] **IONA Technologies PCL**. "OrbixNotification Programmer's Guide and Reference". Dezembro de 1998. http://www.iona.com.

[10] **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.

[11] **Meira, D. M.**. "Um Modelo para Correlação de Alarmes em Redes de Telecomunicações". Tese de doutorado. Belo Horizonte. Novembro, 1997.

[12] **Meira, D. M.**. "Um Survey Sobre Correlação de Alarmes". Technical Report DCC 017/97. Belo Horizonte, Minas Gerais. Julho, 1997.

[13] **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.

[14] **Munich Network Management Team**. "Event Correlation". Novembro, 1998. http://wwwmnmteam.informatik.uni-muenchen.de/projects/evcorr/

[15] **Object Management Group**. "CORBA Services". Março, 1995. http://www.omg.org.

[16] **Object Management Group**. "Notification Service – Joint Revised Submission". Julho, 1998. http://www.omg.org.

[17] **Object Management Group**. "What is CORBA?". http://www.omg.org.

[18] **Object Management Group**. "Management of Event Networks – Request For Proposal". Novembro, 1999. http://www.omg.org.

[19] **Object Management Group**. "Management of Event Domains – Revised Submission". Janeiro, 2000. http://www.omg.org.

[20] **Object Management Group**. "Telecom Log Service – Joint Revised Submission". Maio, 1999. http://www.omg.org.

[21] **Orfali, R., Harkey, D. e Edwards, J.**. "Instant CORBA". John Wiley & Sons. Março, 1997. pp. 115-124.

[22] **Pavón, J**. "Building Telecommunications Management Applications with CORBA". IEEE Communications Surveys. 1999. pp. 2-16.

[23] **PrismTech Limited**. "OpenFusion Developer's Guide". Junho, 1999. http://www.prismtech.com.

[24] **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.

[25] **Rasmussen, B.**. "OpenFusion Notification Service". http://www.prismtech.com.

[26] **Saito, J. T., Madeira, E. R. M.**. "Um Modelo de Gerenciamento de Redes de Telecomunicações Utilizando a Plataforma CORBA". 19°. Simpósio Brasileiro de Redes de Computadores. Florianópolis, Santa Catarina. Maio, 2001. pp. 130-145.

[27] **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.

[28] **WebProForum Tutorials**. "Embedded Telecommunications Management Network (TMN) Solutions". http://www.webproforum.com.

[29] **WebProForum Tutorials**. "Fundamentals of Telecommunications". http://www.webproforum.com.

[30] **WebProForum Tutorials**. "Telecommunication Management Network(TMN)". http://www.webproforum.com.