Serviço Genérico de Log Estruturado para Gerência de Telecomunicações

Ricardo Lovatel de Lemos
rllemos@ppgia.pucpr.br

Sandra Andréa Calixto
sandra@ppgia.pucpr.br

Manoel Camillo Penna
penna@ppgia.pucpr.br

PUC-PR - Pontifícia Universidade Católica do Paraná
Laboratórios de Engenharia Elétrica e Informática
R. Imaculada Conceição, 1155 - Prado Velho - Curitiba - Paraná.
CEP 80215-901

Resumo

Este trabalho apresenta o estudo e especificação de um módulo de software genérico para armazenamento de eventos estruturados . Este módulo chamado de Serviço Genérico de Log Estruturado para Telecomunicações (SGLET) segue o modelo de Log da arquitetura OSP *(Open Systems Interconnection)* e o padrão TMN *(Telecommunications Management Network)* para controle de Log.

O SGLET tem como objetivo principal o armazenamento das mensagens estruturadas ocorridas na rede de telecomunicações, na forma de registros de log, em um banco de dados. Esta aplicação é representada por um módulo para uso em um sistema de gerência de redes e deverá disponibilizar: os registros de log e permitir o controle e manutenção dos logs na rede. Para este fim, é utilizada a linguagem IDL *(Interface Definition Language),* definida pelo OMG *(Object Management Group)* na especificação de uma interface padrão para o Serviço de Log para Telecomunicações.

Abstract

This work presents the study and specification of software module for storage of structured events. This module called Structured Generic Log Service for Telecommunications (SGLET) follows the OSI *(Open Systems Interconnection)* architecture standard for log model, and TMN *(Telecommunications Management Network)* standard for log control.

SGLET main goal is the storage of structured message generated by telecommunication networks, formatted by log records, in a database. This application is represented by a module for using in a network management system, and should: make log records available and allow control and maintenance of logs, in a network. For this purpose, the standard language IDL *(Interface Definition Language)* defined by OMG *(Object Management Group)* is used to specify a standard interface for the Log Service Telecom.

1. Introdução

A gerência efetiva de sistemas de telecomunicações requer que as mensagens relevantes ao funcionamento do sistema (por exemplo, eventos significativos e operações sobre recursos críticos), sejam armazenadas em repositórios de dados, denominados genericamente de Log. O principal objetivo de um log é de preservar as informações sobre os eventos ou operações que possam estar relacionados com equipamentos de telecomunicações gerenciados.

Para gerência dos sistemas de telecomunicações, o ITU-T *(International Telecommunication Union - Telecommunication Standardization Sector)* estabeleceu o

conceito de rede de gerência de telecomunicações (TMN - *Telecommunications Management Network*) [1], que adotou os padrões de gerência OSI *(Open Systems Interconnection)*, como sua base principal. Na gerência OSI, os recursos que armazenam as mensagens críticas do sistema são modelados por dois objetos principais: logs e registros de log [2].

O log definido na gerência OSI possui atributos que estão especificados na recomendação do ITU-T X.735 *"Log Control Function"*. Estes atributos definem as características e o comportamento de um log, e devem ser informados a cada vez que um log é criado. Após a criação, o log estará pronto para receber mensagens, armazenando-as em registros de log. Os atributos do log e os seus registros de log devem estar disponíveis, para manipulação e administração (criação, destruição, consulta e alteração dos atributos e consulta e destruição dos registros de log), para quaisquer outros módulos do sistema de gerência

Este trabalho propõe um Serviço Genérico de Log Estruturado para Telecomunicações (SGLET), para armazenar eventos oriundos da rede de telecomunicações. Os logs gerados por este serviço são estruturados, no sentido que armazenam os registros de log como informação estruturada, isto é, como um conjunto de campos tipados. Esta característica é importante porque permite que os registros de log sejam recuperados, através de comandos de consulta elaborados, que incluem condições de filtragem, construídas sobre o valor de cada um dos campos tipados. O serviço é genérico, no sentido que permite a gerência simultânea de múltiplos logs, que podem armazenar estruturas de informação distintas, permitindo ainda que estes logs sejam introduzidos no serviço (criados), em tempo de execução, sem a descontinuidade do mesmo.

Outra característica importante do SGLET é a sua implementação de acordo com os padrões CORBA *(Common Object Request Broker Architecture)*, desenvolvidos pelo OMG *(Objecí Management Group)*, que definem uma arquitetura para desenvolvimento de aplicações distribuídas baseado na orientação objeto. A construção do SGLET baseado na CORBA é relevante no contexto da TMN, porque, originalmente a intercomunicação em sistemas de gerência segundo as definições da TMN deveria ser realizada por uma interface padronizada Q3. Esta definição inclui uma pilha de protocolos (CMIP - *Common Management Information Protocol*) [3], e um modelo de informação descrito em GDMO *(Guidelines for the Definition of Managed Objects)* [4]. Entretanto, para sistemas de gerência não baseados nos princípios OSI, os padrões TMN também consideram a utilização da arquitetura CORBA para a sua intercomunicação.

Na seção 2 apresentamos o conceito de log para a gerência TMN, ressaltando a influência dos padrões do ITU-T e do OMG na concepção do SGLET. Alguns serviços da CORBA foram usados como base para a concepção e implementação do SGLET, e estão descritos na seção 3. A seção 4 apresenta o serviço de log proposto, incluindo sua concepção, e a relação com os serviços que lhe servem de apoio. A seção 5 descreve alguns aspectos de implementação, e a seção 6 conclui o trabalho.

2. Conceito de Log para Gerência TMN

Em muitas aplicações não padronizadas o log é um repositório de dados simples, muitas vezes implementado como um *buffer*, um arquivo ou um sistema de arquivo. Porém em aplicações de gerência de redes, a quantidade de eventos que são gerados aumenta conforme a complexidade da rede, sendo necessário um sistema mais elaborado para realizar estafunção.

2.1 Função de Controle de Log do ITU-T

Para preservar as informações deste eventos ocorridos em um rede de equipamentos de telecomunicação e permitir o acesso a estas informações de modo padronizado, o ITU-T

modelou, na recomendação X.735, estes repositórios de dados através de um objeto denominado log, e os eventos como registros de log. A recomendação X.735 define as funcionalidades e características que o log deve possuir, mas não estabelece a forma de implementação do log.

A classe log é caracterizada por um pacote de atributos obrigatório e vários pacotes condicionais. A seguir será apresentado uma breve explicação do pacote obrigatório e de alguns dos pacotes condicionais que foram utilizados na concepção do SGLET.

Os atributos que fazem parte do pacote obrigatório são:

- *Log* ***id***: Identifica unicamente uma instância de log.
- *Discriminator construct:* Filtra as informações a serem armazenadas.
- *Administrative* ***state***: Define a capacidade administrativa de funcionamento do log. Possui dois estados administrativos: ***UNLOCK*** (o log está disponível para armazenar, recuperar e remover registros de log), e *LOCK* (o log está disponível apenas para a leitura e remoção de registros de log).
- *Operational state:* Representa a capacidade operacional do log. Possui dois estados operacionais: ***ENABLE*** (o log está criado e pronto para uso), e ***DISABLE*** (o log não está disponível para uso).
- *Log full* ***action***: Define a ação a ser tomada quando a condição de log cheio *(log full)* acontece. Possui dois modos de operação: ***WRAP*** (os registros mais antigos do log são removidos liberando espaço para os registros mais novos), e *HALT* (nenhum novo registro será colocado no log, sendo que os registros mais novos são descartados e os mais antigos conservados).
- ***Availability status***: Determina a disponibilidade do log. Este atributo é utilizado por inúmeras outras classes de objeto da TMN. No escopo do objeto log, este atributo pode indicar uma condição de log cheio, que impede que novos registros de log sejam criados.

Três atributos fazem parte do pacote condicional tamanho de log finito *(Finite log size* ***package***), que inclui as informações relacionadas ao tamanho do log:

- ***Max*** *log size:* Define o tamanho máximo do log em número de bytes. Se assumir valor zero indica que o log possui tamanho indeterminado.
- *Current log size:* Especifica o tamanho corrente do log em bytes.
- ***Number*** *of* ***records***: Especifica o número corrente de registros contidos no log.

Um único atributo faz parte do pacote condicional alarme de log *(Log alarm* ***package***), que permite informar quando uma condição de log cheio aproxima-se (este pacote é obrigatório para um log que permita o a opção ***halt*** no atributo *log full action);*

- *Capacity alarm threshold:* Especifica, como uma porcentagem do tamanho máximo do log, os níveis de capacidade a partir da qual será gerada uma condição de aproximação de log cheio. Estes limites deve estar entre O e 100%.

A classe registro de log representa a informação armazenada no log. Um registro de log é criado quando o log recebe um evento, e possui os seguintes atributos obrigatórios, que foram utilizados na concepção do SGLET:

- *Log record id:* Identifica o registro de log unicamente no escopo do log.
- ***Logging*** *time:* Indica o instante em que o registro de log foi armazenado no log.

Segundo a especificação X.735, para que um log seja criado alguns de seus atributos devem ser definidos no momento de criação. Quando o valor de um destes atributos não é fornecido, assume-se um valor *default*. Os atributos necessários para criar um log são: *max log* ***size***; *capacity alarm threshold; log full action; discriminator construct* e *administrative state*.

A principal função de um log é armazenar os registros de log. As seguintes condições devem ser satisfeitas para que um registro de log seja armazenado, quando uma mensagem chega ao log:

- A mensagem deve satisfazer a condição de filtro definida no atributo ***discriminator construct.***
- Os valores dos atributos ***administrative*** *state* e ***operational state*** forem iguais a ***unlocked,*** *enable,* respectivamente.
- O valor do atributo ***availability status*** não for igual a *log**full.***

Quando um log atinge sua capacidade máxima, o seu comportamento pode ser de acordo com uma das duas possibilidades a seguir, dependendo do valor do atributo *log **full** action:*

- Se o valor do atributo for ***halt,*** ele gerará uma notificação do tipo ***capacityalarm*** *threshold,* indicando que a capacidade máxima foi atingida, e refletirá esta condição no atributo *availability status,* alterando seu valor para ***log full.***
- Se o valor do atributo for ***wrap,*** ele armazenará os novos registros de log sobre os registros de log mais antigos.

Durante a sua execução, um log deverá emitir notificações para informar os eventos mais relevantes. As seguintes notificações devem ser geradas:

- ***ObjectCreation***: Esta notificação é gerada quando uma nova instância do log é criada. A notificação de *ObjectCreation é* definida na recomendação X.730 **[5].**
- ***ObjectDeletion***: Esta notificação é gerada quando uma instância do log é removida. A notificação de *ObjectDeletion* é definida na recomendação X.730 **[5].**
- ***AttributeValueChange***: Esta notificação é gerada quando uma ação de gerência altera algum dos seguintes atributos: *capacity alarm* ***threshold,*** *discriminator construct, log full action,* ***max*** *log size* (este valor não pode ser alterado por um valor menor que o *current log size).* A notificação de *AttributeValueChange* (definida na recomendação X.730 **[5])** informa o atributo alterado, seu valor antigo e seu novo valor.
- ***StateChange***: Esta notificação é gerada quando ocorre uma alteração no atributo *operational state* ou quando uma ação de gerência altera o estado do atributo *administrative state*. A notificação de *StateChange* (definida na recomendação X.731 **[6])** informa o atributo alterado e seu novo estado.
- ***ProcessingErrorAlarm***: Esta notificação é gerada quando algum dos níveis definidos no atributo *capacity alarm threshold* é atingido. A notificação de *ProcessingErrorAlarm* (definida na recomendação X.733 **[7])** informa a severidade, crítico se o valor atingido for 100% da capacidade *(log **full**)* e menor para outros valores, e o valor de capacidade atingido.

2.2 Serviço de Log para Telecomunicações do OMG

O Serviço de Log para Telecomunicações do OMG *(Telecom Log Service)* [8], é a versão CORBA da recomendação X.735. O serviço de log do OMG introduz entretanto diversas extensões:

- A possibilidade de formação de redes de log, onde os eventos recebidos em um log podem ser repassados para outros logs.
- A definição do conceito de qualidade de serviço (QoS) com propriedades aplicadas somente aos logs.
- A definição de uma linguagem para a filtragem e pesquisa de registros de log, a *Constraint **Language,*** definida no Serviço de Notificação **[9].**

As mensagens podem chegar ao log de duas maneiras distintas em CORBA, a maneira usual dos objetos de um serviço interagirem com os demais, é através da especificação de uma interface própria (em IDL - *Interface Definition Language*), onde estarão definidas as operações que podem ser invocadas para realizar a interação e a troca de mensagem. Assim, a interface log define as operações write_records() ou write_recordlist(). A invocação de uma operação em CORBA é sempre realizada de modo síncrono, exigindo que os objetos que interagem estejam simultaneamente em execução.

Para permitir a interação entre objetos de forma não síncrona, o OMG definiu um serviço de eventos, que define um objeto intermediário denominado canal de evento [10]. O serviço de log prevê que a comunicação com o log possa se realizar através de um canal de evento. A comunicação com o log poderá ser feita através de um canal de notificação, que é uma extensão do canal de evento. Um log do SGLET recebe suas mensagens através de um canal de evento, que será apresentado na seção 3.1.

Para permitir o reuso do canal de evento (ou notificações), o log deve herdar de um deles, conforme ilustrado na *Figura 1*. Observa-se ainda que o canal pai na hierarquia de herança, pode ser tanto tipado como não tipado, o que significa que as mensagens que chegam ao log são estruturadas em campos tipados ou não. Esta característica é relevante ao contexto deste trabalho, e uma das suas contribuições é apresentar um mecanismo genérico para que mensagens tipadas possam ser tratadas por um log conectado a um canal de evento não tipado. Esta contribuição é relevante porque as atuais implementações do canal de evento são não tipadas.

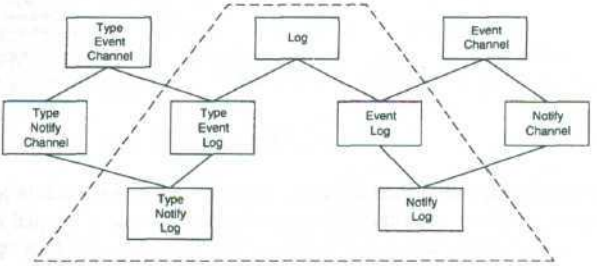

Figura 1 - Herança de Log

A Figura 1 de herança de log apresenta os tipos de interfaces de log que podem existir e sua hierarquia.

- *Log:* A interface *Log* serve como uma interface abstrata a partir da qual todas as outras interfaces de log herdam. *Log* define atributos e operações comuns para todas as outras interfaces de log.
- ***EventLog***: Esta interface herda do *Log* as operações e atributos de log, e herda da interface ***CosEventChannelAdmin::EventChannel*** pela qual recebe e repassa (para outros logs) eventos genéricos.
- ***NotifyLog***: Esta interface herda de ***EventLog*** e da interface ***CosNotifyChannelAdmin::EventChannel***, pela qual recebe e repassa eventos genéricos. A interface *NotifyLog* permite a filtragem de eventos genéricos, tanto para armazenamento como para repasse, e a utilização da QoS definida no Serviço de Notificação
- ***TypedEventLog***:Possui as mesmas características da interface *EventLog*, porém é utilizada para eventos tipados. Esta interface herda de *Log* e da interface ***CosTypedEventChannelAdmin::TypedEventChannel*** pela qual recebe e repassa

eventos tipados.

- ***TypedNotifyLog*** Possui as mesmas características da interface *NotifyLog*, porém é utilizada para eventos tipados. Herda de ***TypedEventLog*** e da interface ***CosTypedNotifyChannelAdmin::TypedEventChannel*** pela qual recebe e repassa eventos tipados.

Além das interfaces mostradas na Figura 1, existe a interface ***BasicLog*** que herda diretamente da interface abstrata **Log.** Esta interface permite que clientes acessem o **log** diretamente sem conhecimento dos eventos ou notificações. Não suporta repasse de eventos nem emite eventos de log.

Para cada uma destas interfaces de log, o serviço de log define uma *log* ***factory*** correspondente, conforme mostra Figura 2. Uma *log factory é* uma especialização de uma *factory,* a interface padrão para controle de ciclo de vida de objetos em CORBA. Ela é responsável por criar instâncias de log, emitir os eventos gerados por estas instância para aplicações (consumidores) interessadas e localizar instâncias de log. Desta maneira as instâncias de log das interfaces, *BasicLog,* ***EventLog,*** *NotifyLog, TypedEventLog* e *TypedNotifyLog* são criadas pelas suas *log factories* respectivas, ***BasicLogFactory, EventLogFactory, NotifyLogFactory, TypedEventLogFactory, TypedNotifyLogFactory.***

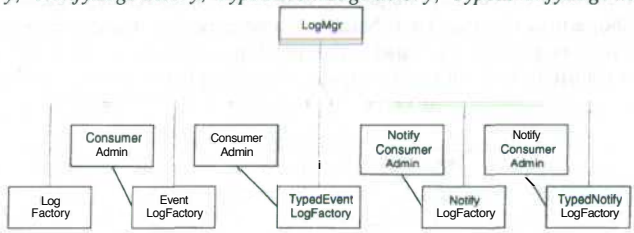

Figura 2 - Herança de Factory

O serviço de log prevê duas maneiras para criar uma instância de log utilizando sua *log factory* correspondente. A primeira utiliza a operação **create(),** na qual deve-se passar os parâmetros *log* ***full action,*** *max log size, capacity* ***alarm threshold.*** Esta operação retorna o **identificador** da instância criada. A segunda maneira utiliza a operação create_with_id(), que permite passar como argumento o valor do identificador da instância de log a ser criada, além dos parâmetros anteriores.

Em tempo de execução, um **log** pode gerar diversos eventos para efeito de gerência. Os eventos definidos pelo serviço de log do OMG são: *ObjectCreation Event, ObjectDeletion Event,* ***ThresholdAlarm*** *Event (Processing Error Alarm),* ***AttributeValueChange*** *Event, StateChange Event.* As estruturas destes eventos são mostradas na Figura 3.

```
struct ObjectCreation (
    Log logref;
    LogId id;
    TimeT time;
);

A estrutura do ObjectDeletion é
a mesma do ObjectCreation
```

```
struct ThresholdAlarm (
    Log logref;
    LogId id;
    TimeT time;
    Threshold crossed_value;
    Threshold observed_value;
    PerceivedSeverity perceived severity;
);
```

```
struct AttributeValueChange (
    Log logref;
    LogId id;
    TimeT time;
    AttributeType type;
    any old_value;
    any new value;
);
```

```
struct StateChange {
    Log logref;
    LogId id;
    TimeT time;
    StateType type;
    any new value;
};
```

Figura 3- Estruturas dos eventos gerados pelo log.

3. Serviços OMG Usados na Construção do SGLET

Para o desenvolvimento do SGLET foi utilizado o Serviço de Evento e o Repositório de Interface, ambos definidos pelo OMG, e são apresentados resumidamente a seguir.

3.1 Serviço de Evento

A principal finalidade do serviço de evento [10] é desacoplar a comunicação entre objetos, permitindo a troca de mensagens assincronamente. Neste modelo, são definidos dois papéis distintos para os objetos: o fornecedor *(supplier),* que produz as mensagens (chamadas de eventos) e o consumidor *(consumer),* que recebe os eventos gerados. Tanto fornecedores como consumidores se conectam a um outro objeto chamado de Canal de Evento, que funciona como um objeto mediador. O canal de evento é tanto um consumidor como um fornecedor de eventos, possibilitando a comunicação de múltiplos fornecedores e consumidores assincronamente, sem que estes precisem saber da existência uns dos outros.

Os objetos *proxy* desempenham um importante papel dentro do canal de evento. Eles são utilizados tanto por fornecedores como consumidores para que estes possam se conectar ou desconectar do canal de evento. Além disso, é através destes objetos que fornecedores enviam eventos e consumidores recebem eventos.

São definidos dois modelos de comunicação de eventos: o modelo *Push,* onde o fornecedor toma a iniciativa de transmitir eventos para o canal de evento, e este por sua vez transmite os eventos para os consumidores (Figura 4), e o modelo *Pull,* onde o consumidor requisita por um evento ao canal de evento e este por sua vez pergunta ao fornecedor se ele tem algum evento para ser enviado (Figura 5). Um fornecedor pode estar utilizando uma determinada modalidade de comunicação, por exemplo o modelo *push,* enquanto o consumidor pode estar utilizando outra modalidade, por exemplo o modelo *pull,* para se comunicarem.

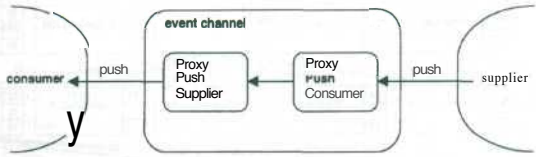

Figura 4 - Modelo de comunicação push

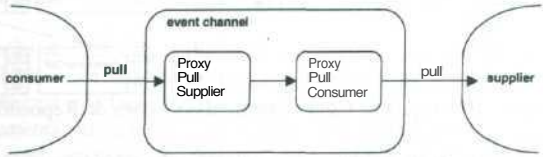

Figura 5 - Modelo de comunicação pull

Conforme veremos na seção 5.1, o log do SGLET conecta-se a um canal de evento como consumidor para receber os registros de log, e como fornecedor, para transmitir os eventos gerados internamente (por exemplo, *log full).* Para ambos os casos foi usado o modelo *push.*

3.2 Repositório de Interface

Para representação de tipos de dados, a CORBA define os códigos de tipo *(typecodes)*

que correspondem à representação de cada tipo de dado que pode ser definido em **IDL**. Estes códigos de tipo são utilizados para permitir o mapeamento entre diferentes sistemas operacionais ou diferentes implementações de ORBs *(Object Request Broker)*. Além do código de tipo, as outras estruturas lingüísticas da IDL também possuem uma representação computacional. Isto é o que chamamos de ***metadado***.

O Repositório de Interfaces (RI) [l 1] é uma base de dados que contém um dicionário de metadados, que representam as **IDLs**. As estruturas de dados que representam as estruturas lingüísticas da IDL podem ser recuperadas do RI, identificando os tipos definidos em tempo de execução. O principal uso do RI é a recuperação em tempo de execução da estrutura lingüística de uma interface, permitindo a construção dinâmica de uma invocação. Este enfoque dinâmico contrapõe-se ao mecanismo de interação mais tradicional, denominado de enfoque estático, que é baseado na compilação das IDLs para a criação de ***stubs***, que serão usadas nas interações.

O conteúdo do RI deve refletir as interfaces definidas em IDLs. Os elementos (objetos) que um RI pode conter são portanto, as definições que podem existir em IDL, isto é módulos, interfaces, operações, parâmetros, exceções, tipos e constantes. São definidos então sete classes de objetos para representar estas definições: ***ModuleDef***, ***InterfaceDef***, *OperationDef*, ***ParameterDef***, ***ExceptionDef***, ***ConstantDef***, e *TypeDef*. Além destes, o objeto ***Repository*** representa o próprio RI. Na Figura 6 [12], os classes de objetos que podem existir no RI, são apresentadas, organizadas através de uma relação de conteúdo. Isto é, a figura indica que um objeto da classe *Repository* pode conter objetos das classes *ConstantDef, Typedef, ModuleDef, ExceptionDef,* e *InterfaceDef,* e assim sucessivamente.

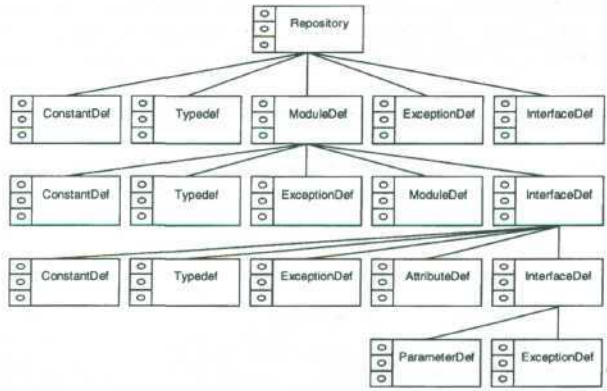

Figura 6 - Hierarquia de Contenimento das Classes do Repositório de Interfaces.

Baseado nesta relação de conteúdo existente entre as classes de objetos do RI, foram definidas três classes abstratas chamadas *IRObject, Contained* e *Container*. Todos os objetos pertencentes ao RI herdam da interface de *IRObject,* que possui um atributo para identificar o tipo de objeto e um método para destruir o objeto. Objetos que "contêm" outros objetos herdam da interface *Container* e objetos "contidos" em outros objetos herdam da classe *Contained*. Na Figura 7 [12] temos uma representação da relação de herança entre estas classes abstratas, incluindo ainda o detalhe das duas classes mais importantes: *InterfaceDef e Repository*. Todas as demais classes herdam destas classes.

Conforme será visto na seção 5.2, o **log** do SGLET utiliza o RI para recuperar dinamicamente e estrutura do campo de informação dos registros de um log particular,

permitindo que logs especializados possam ser criados pelo SGLET em tempo de execução.

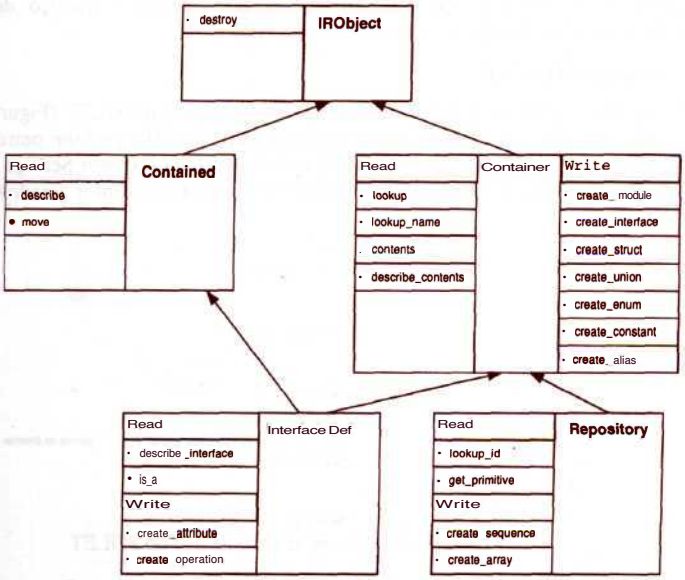

Figura 7 - Hierarquia de Classes do Repositório de Interfaces

4. Serviço Genérico de Log Estruturado para Telecomunicações • SGLET

O SGLET é um módulo de software que tem como objetivo principal o armazenamento de eventos, operações ou ações ocorridas em uma rede de equipamentos de telecomunicações, na forma de registros de log, sendo também responsável por, criar e remover instâncias de log; consultar e alterar os atributos de uma instância particular de log; criar, remover e consultar registros de log; emitir eventos de criação e remoção de instâncias de log, de alteração de atributos de uma instância de log; e emitir alarmes de capacidade do log.

Para a concepção do SGLET, utilizou-se a recomendação do ITU-T X.735, para definição de seus atributos e de seu comportamento, e a especificação Serviço de Log do OMG, para definição de suas interfaces IDLs.

Além de implementar as funções padronizadas pelo ITU-T, em um componente de software compatível com um serviço definido pelo OMG, o SGLET estende estas especificações em funcionalidade. A principal extensão é a possibilidade do log armazenar eventos estruturados, possibilitando que logs distintos, que armazenam estruturas distintas, possam ser instanciados em um mesmo servidor, dinamicamente, sem necessidade de descontinuidade de execução, isto é, um novo log, que armazena eventos com uma nova estrutura, pode ser instanciado sem tirar o serviço do ar.

Estas extensões são importantes, pois permitem que o serviço de log crie novos repositórios (logs) em tempo de execução. Além disso, devido ao fato que o conteúdo da informação armazenada nos registros de log serem tipados, é possível que estes registros possam ser filtrados em tempo de consulta, através do uso de linguagens de *query* elaboradas,

como por exemplo a SQL.

A seguir serão apresentados alguns detalhes do projeto de software do SGLET, documentado usando-se a UML [13]. Discute-se ainda como o Serviço de Evento e Repositório de Interface foram reusados na sua concepção.

4.1 Concepção do SGLET

A seguir é apresentado o diagrama de caso de uso para o SGLET (Figura 8). O ator cliente é uma entidade externa que representa uma interface gráfica ou um outro sistema de gerência interessado em adquirir ou administrar os logs existentes. O ator Serviço de Evento é o responsável por enviar os eventos para o SGLET, e repassar os eventos gerados para outros consumidores.

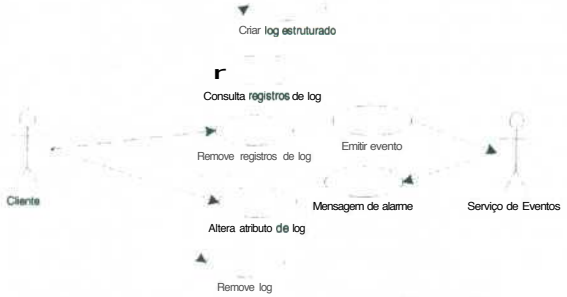

Figura 8 - Diagrama de caso de uso do SGLET

São apresentados a seguir os diagramas de estado das duas principais ações do log, acionadas após um comando de criação *(Figura* 9), e após a chegada de uma mensagem *(Figura* 70).

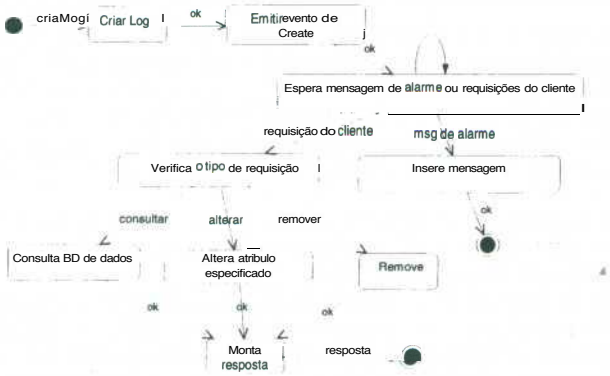

Figura 9 - Criação de uma instância de log.

Na Figura 11 é apresentado o modelo de objetos para o desenvolvimento do SGLET, mostrando as classes herdadas necessárias. O SGLET estende as interfaces *EventLog* e *EventLogFactory*. A classe *EventGenericLogFactory* utiliza a classe *PushSupplier_i* (esta estende a interface *PushSupplier*) para emitir eventos gerados pelos log (instância do *EvetGenericLog*). A classe *EventGenericLog* utiliza a classe *PushConsumer_* (esta estende a

interface *PushConsumer*) para receber os eventos que chegam a um determinado Canal de Evento.

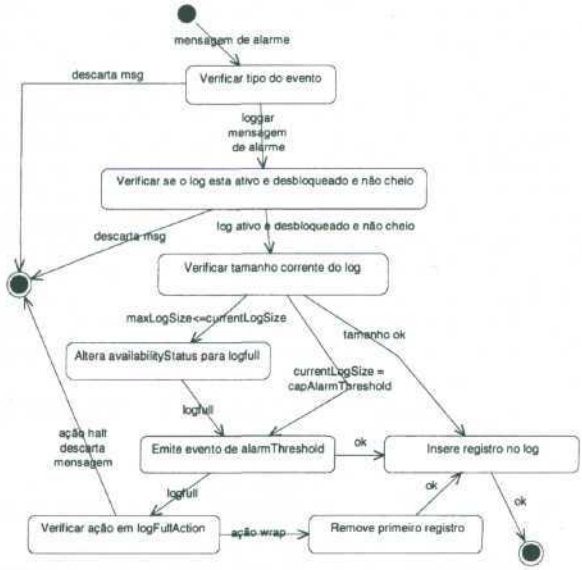

Figura 10 - Comportamento de uma instância particular de log.

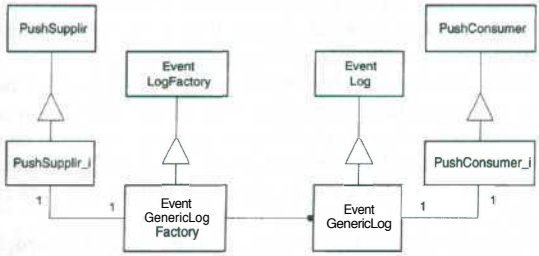

Figura 11 - Interfaces utilizadas no desenvolvimento do Log Genérico.

Detalhes da documentação de projeto do SGLET, incluindo os diagramas de casos de uso, de classes, de seqüência e de estado, podem ser encontrados em [14].

4.2 Utilização dos Serviços OMG

Conforme dito na seção 3.1, o SGLET usa um canal de evento (CE) para receber os registros de logs, e para emitir as suas notificações. Segundo a especificação do OMG, um CE suporta estilos de comunicação de eventos genéricos e tipados. Comunicação de eventos genéricos envolve a transmissão das mensagens em um fluxo de bytes não tipado (formato *any*). A comunicação envolvendo eventos tipados envolve a transmissão de mensagens de estrutura pré estabelecida entre os fornecedores e consumidores através de uma interface definida em IDL.

O Serviço de Notificação introduziu um novo estilo de mensagem de evento,

denominado Evento Estruturado (*Structured Event*), que é uma estrutura de dados bem definida, capaz de mapear diversos tipos de eventos. Esta estrutura de dados possui uma parte do cabeçalho fixa com três campos (*domain_name*, *type_name* e *event_name*). O conteúdo do evento é composto por pares nome e valor, onde o nome é do tipo *string* e o valor é no formato *any*.

Na concepção do SGLET foi adotada uma combinação destas possibilidades: utilizamos o formato do evento estruturado do serviço de notificações, sendo transmitido através de um canal de evento não tipado, com a estrutura sendo acordada entre o fornecedor do evento e o log, e registrada no RI para tratamento dinâmica. Esta opção justifica-se com os seguintes argumentos. O canal de notificação não foi usado, porque é uma estrutura muito pesada para comunicação de eventos, e contém funções não necessárias para um serviço de log. Além disso, no início da concepção do SGLET, o serviço de notificação ainda não era um padrão estabelecido. O canal de evento foi utilizado, porque a comunicação não síncrona é o melhor modelo para eventos, e além disso, o canal de evento é um padrão estabelecido, com diversas implementações disponíveis. O mecanismo de identificação dinâmica através do RI foi incluído, pois entendemos que a facilidade de eventos estruturados é fundamental para o serviço de log.

Segundo a especificação do Serviço de Log do OMG, os registros de log, não tipados, possuem o seguinte formato:

```
struct LogRecord{
        RecordId        id;
        TimeT           time;
        NVList attr_list; // esta informação não faz parte do evento
        any             info;
};
```

A consulta aos registros de log é realizada através do método *retrive()*, definido nesta especificação. Neste método, pode-se apenas selecionar o número de registros desejados, a partir de um instante *t*. Não é possível se estabelecer nenhum critério de consulta que seja baseado no conteúdo da informação do evento. Esta advém principalmente do fato que as informações específicas do evento estão no formato *any*.

Para possibilitar a consulta dos registros de log incluindo as facilidades existentes linguagem de consulta, a informação do evento deve ser armazenada em campos estruturados. No SGLET a estrutura do campo de informação é definida em tempo de criação de um novo log, através de uma estrutura definida em IDL e cadastrada no RI. No momento da criação de um log, é informado ao servidor de log, o nome da estrutura que será recuperada do RI na criação do novo log.

Para permitir a criação de um log estruturado em tempo de execução, a IDL de criação de log do serviço OMG foi estendida. Uma nova operação de criação foi incluída na interface, contendo um parâmetro para especificar o nome da estrutura pré-cadastrada no RI que será utilizada para definir a estrutura dos registros de log [15]. A nova operação para criação de log, é definida em IDL como segue:

```
EventLog create_with_name (
        in DsLogAdmin::LogFullActionType full_action,
        in unsigned long max_size,
        in DsLogAdmin::CapacityAlarmThresholdList thresholds,
        in string event_structured_name,
        out DsLogAdmin::LogId id
)
raises (DsLogAdmin::InvalidThreshold);
```

Quando esta operação é invocada, o parâmetro *event_structured_name* contém nome da estrutura que será usada pelo SGELT para montar uma tabela interna, que armazenará os eventos do novo log. Esta estrutura definirá a estrutura dos registros de log. Os passos seguidos na criação de um novo log estruturado pelo SGLET são apresentados a seguir, e

ilustrados na Figura 12:

1. A IDL que contém a estrutura do evento a ser armazenado no novo log é incluída no Repositório de Interfaces.
2. A operação criação de um log estruturado é invocada, passando o nome da estrutura através do parâmetro *event_structured_name*.
3. O SGELT procura a estrutura no repositório de interfaces, e obtém o nome e tipo de cada componente.
4. Com o formato da estrutura, o SGELT realiza o mapeamento dos tipos IDL definidos na estrutura e cria uma expressão SQL, para a criação da tabela de armazenamento de registros de log.
5. O banco de dados recebe o comando SQL e executa.
6. Ao receber a confirmação de sucesso do banco de dados, SGELT cria um objeto log.
7. O SGELT envia para o canal de evento, um evento de *ObjectCreation*. Este evento indica que foi criado um novo objeto log, e este está pronto para armazenar os eventos do tipo especificado.

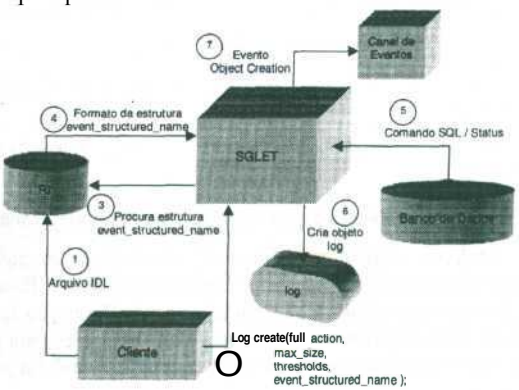

Figura 12- Processo de criação de log

Após a criação o log está pronto para receber os eventos. Para que seja possível identificar os eventos que chegam no log, e verificar se estes eventos devem ou não ser armazenados, o campo *type_name* do evento estruturado é comparado com o nome da estrutura utilizada para criação. Portanto, o nome da estrutura especificada deverá possuir o mesmo nome do tipo do evento a ser armazenado.

5. Aspectos de Implementação

De modo a validar o conceito de Serviço Genérico de Log Estruturado, foi desenvolvido um protótipo, sobre a plataforma Windows NT, utilizando o ORB omniBroker (oficialmente Orbacus) e o JDK 1.2.2 (*Java Development Kit*). O acesso ao banco de dados relacionai foi realizado utilizando o driver Oracle JDBC (*Java Database Connectivity*), que utiliza a especificação JDBC 2.0.

Esta seção apresenta alguns detalhes de implementação, incluindo a interação com o canal de evento, e a interação com o repositório de interface.

5.1 Interação com o Canal e Eventos

No processo de conexão com o Canal de Evento o consumidor inicialmente obtém do Serviço de Nomes uma referência para o objeto do Canal de Evento (Exemplo 1, linha 4), sendo este um objeto do tipo *Object*. Esta referência precisa ser ajustada para ser do tipo *EventChanel* (Exemplo 1, linha 17), através do método *narrow()* fornecido *EventChanelHelper* , como mostrado no exemplo a seguir:

```
1. org.omg.CORBA.Object event_channel_object_ = null;
2. try
3. {
4.     event_channel_object_ = orb_.resolve_initial_references("EventService");
5. }
6. catch(org.omg.CORBA.ORBPackage.InvalidName ex)
7. {
8.     System.err.println("Can't resolve 'EventService'");
9.     System.exit(1);
10.}
11. if(event_channel_object_ == null)
12.{
13.    System.err.println("'EventService' is a nil object reference");
14.    System.exit(1);
15.}
16.
17. Eventchannel eventchannel = EventChannelHelper.narrow(event_channel_object_);
18. if (null == eventChannel)
19.{
20.    System.out.println("Consumer: can't create Event Channel from IOR");
21.    System.exit(1);
22.)
```

Exemplo 1 - Obtenção uma referência para o objeto canal de evento

Tendo-se obtido o objeto *eventChannel*, deve-se então instanciar um objeto *pushConsumer*, fornecendo como argumento o objeto *eventChannel* (Exemplo 2, linha 24). O método *connect()* do objeto *pushConsumer* é então invocado (Exemplo 2, linha 25).

O consumidor deve implementar a interface *PushConsumer* para permitir que o canal de evento possa desconectar o fornecedor quando necessário e também para poder receber os eventos.

```
23. ConsumerDataHandler dataHandler = new ConsumerDataHandler();
24. PushConsumer_i pushConsumer_ = new PushConsumer_i (eventChannel, dataHandler);
2B.pushConsumer_.connect();
26.
27. // Enter in the event loop
28. System.out.println("Consumer: Waiting for events");
29 . boa.impl_is_ready(null);                                          t
```

Exemplo 2 - Conexão de um consumidor ao canal de evento

Para conectar um fornecedor no canal de evento, basta substituir as linhas 23 a 25 do Exemplo 2, pelas linhas 30 e 31 do Exemplo 3.

```
30. pushSupplier_ = new PushSupplier_i (eventChannel);
31.pushSupplier_.connect();
```

Exemplo 3 - Conexão de um fornecedor ao canal de evento

5.2 Interação com o RI

O código apresentado no Exemplo 4, mostra o procedimento para obter as informações armazenadas no RI que são necessárias no funcionamento do SGLET.

Para encontrarmos a estrutura cadastrada previamente, fornecemos o objeto Repositório de Interface para o método *initi (Exemplo 4,* linha 1). Com este objeto, invocamos uma operação que permite verificar o seu conteúdo (*contents*). A resposta (*contained*) é uma lista que inclui com os objetos diretamente contidos por este objeto (Exemplo 4, linha 3). Então inicia-se o laço para encontrar todos os tipos existentes dentro deste repositório (Exemplo 4, linha 4). Ao encontrarmos o módulo definido (Exemplo 4, linha 6), começa-se a procurar os tipos definidos dentro deste módulo (Exemplo 4, linha 7). Uma vez encontrada uma estrutura, deve-se verificar se o nome da estrutura, é o mesmo fornecido na criação do Log. A seguir se entra novamente nesta função para descobrir cada tipo contido dentro desta estrutura.

```
1. private String initi(org.omg.CORBA.Container container) {
2.   String texto="";
3.   org.omg.CORBA.Contained[] contained =
       container.contents(org.omg.CORBA.DefinitionKind.dk_all, true);
4.   for(int i = 0; i < contained.length; i++) {
5.     switch(contained[i].def_kind().value()) {
6.       case org.omg.CORBA.DefinitionKind._dk_Module: {
7.         this.initi(org.omg.CORBA.ModuleDefHelper.narrow(contained[i]));
8.         break;
9       }
10.      case org.omg.CORBA.DefinitionKind._dk_Struct: {
11.        texto=printStruct(org.omg.CORBA.StructDefHelper.narrow(contained[i]));
12.        break;
13.      }
14.      default:
15.        break;
16.    }
17.  }
18.  return texto;
19.}
20.//------------------------- printStruct() ----------------------------
21.private void printStruct(org.omg.CORBA.StructDef def,
     String StructName) {
22.  if (def.name().equals(StructName)) {
23.    println("struct " + def.name() + " {");
24.    indent++;
25.    org.omg.CORBA.StructMember[] members = def.members();
26.    for(int j = 0; j < members.length; j++) {
27.      println(toIdl(members[j].type_def) + " " +
28.      members[j].name + ";");
29.    }
30.    indent--;
31.    println("};");
32.  }
33.}
```

Exemplo 4 - Código para varrer o Repositório em busca de um tipo específico

6. Conclusão

Este trabalho apresentou um Serviço de Genérico de Log Estruturado para Gerenciamento de Redes de Telecomunicações, que tem como principais características, a possibilidade de armazenar registros de log de forma estruturada, que permite definir dinamicamente novos logs, para novos tipos de registros de log.

O conceito foi validado através da implementação de um módulo de software, utilizado como um dos componentes de uma plataforma de supervisão de falhas [16]. Esta implementação foi testada, e o seu desempenho foi considerado satisfatório.

Uma das principais vantagens observadas, foi que os registros de log ficam armazenados em tabelas relacionais, o que possibilita a sua consulta através de linguagens de *query* poderosas. Particularmente, foi implementada um aplicação de consulta baseada em SQL.

Diversas extensões estão em estudo para o SGLET:

- Criação de uma API de consulta baseada em CORBA.
- Criação de uma API para registro de interesse em eventos, incluindo facilidade de filtro e qualidade de serviço, agregando ao SGLET as funções de um serviço de telecomunicações.

7. Agradecimentos

Agradecemos a Siemens pelo suporte dado a este trabalho que faz parte do Projeto GIR (Gerência Integrada de Redes) desenvolvido em cooperação entre a Siemens e a PUC-PR. Este projeto é coordenado na Siemens pelo departamento ICN TRD.

8. Referências Bibliográficas

[1] ITU-T Rec. M.3010 - Principies for a Telecommunications Management Network, 1996.

[2] CCITT. Rec. X.735 - Information Technology - Open Systems Interconnection - Systems Management: Log Control Function, 1992.

[3] CCITT. Rec. X.711 - Common Management Information Protocol Specification For CCITT Applications, 1991.

[4] CCITT. Rec. X.722 - Information Technology - Open Systems Interconnection - Systems Management Information: Guidelines For The Definition Of Managed Objects, 1992.

[5] CCITT. Rec. X.730 - Information Technology - Open Systems Interconnection - Systems Management: Object Management Function, 1992.

[6] CCITT. Rec. X.731 - Information Technology - Open Systems Interconnection - Systems Management: State Management Function, 1992.

[7] CCITT. Rec. X.733 - Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function, 1992.

[8] Documento OMG, "Telecom Log Service", telecom/99-05-01. Disponíveis em http://www.omg.org/pub.

[9] Documento OMG, "Notification Service", telecom/99-07-01. Disponíveis em http://www.omg.org/pub.

[10] Documento OMG,, "CORBAService: Common Object Service Specification", ed. rev. Dezembro 1998. Disponíveis em http://www.omg.org.

[11] Documento OMG, "CORBA: Common Object Request Broker Architecture and Specification", ed rev. Fevereiro 1998. Disponíveis em http://www.omg.org.

[12] ORFALI, Robert; HARKEY, Dan ,"Client/Server Programming with JAVA and CORBA". John Wiley & Sons, Inc. 2nd ed, 1998.

[13] FURLAN, José Davi, "Modelagem de Objetos através da UML - The Unified Modeling Language". Makron Books, 1998.

[14] LAGIR-PUC/PR. "Documento de Análise e Projeto dos Módulos Acesso ao NE e Serviço Genérico de Log Estruturado", Relatório Técnico. PUC-PR, agosto 1999.

[15] LAGIR-PUC/PR. "Uso do Repositório de Interface como Interface entre o Serviço de Eventos e o Log Genérico", Relatório Técnico. PUC-PR, dezembro 1999.

[16] LAGIR-PUC/PR. "Plataforma de Supervisão de Alarmes Baseada em CORBA", artigo submetido ao SBRC 2000. PUC-PR, fevereiro 2000.