Implementação de Gerenciamento de Serviços de Telecomunicações utilizando CORBA, Java e Reflexão Computacional

Paulo Roberto Riccioni Gonçalves
riccioni@sj.univali.rct-sc.br
Grupo de Pesquisa em Tecnologia e Sistemas
Departamento de Ciência da Computação
Universidade do Vale do Itajaí - UNIVALI - Campus VII
Rod. SC 407 , Km 4 - 88122-000-São José - SC

João Bosco Mangueira Sobral
bosco@lrg.ufsc.br
Laboratório de Rede e Gerência
Departamento de Informática e de Estatística
Universidade Federal de Santa Catarina
Cx.Postal 476 - 88040-900-Florianópolis-SC

Abstract

Currently, the telecommunications management systems are powerful, however, maintain centralized and proprietary platforms, limiting this way, the growth of these telecommunications networks. This factor generates the need for scalable and open systems, stimulating the sectors of the telecommunications area to seek for solutions for the problem in distributed, heterogeneous and totally scalable environments. The recent advances in the area of distributed objects, has created many positive expectations for the development of client/server systems which use, as if they were local, resources available through the network. This way, the objective of this work consists in the experimental evaluation of the potential of the main CORBA services in conjunction with the Java language, through the necessary requirements for the main functionalities proposed by the TMN architecture. For the implementation of the telecommunication services with the reusability of codes we use computational refection with meta-classes substituting the manager and classes substituting the agents

Keywords: Distributed Systems, Distributed Objects, Computational Reflection, CORBA, JAVA, TMN

1. Introdução

Os recentes avanços na área de **objetos distribuídos** têm permitido muitas expectativas positivas para o desenvolvimento de sistemas cliente/servidor que utilizam, como se fossem locais, recursos disponíveis através da rede, utilizando toda a conceituação de objetos e componentes de software. A *Internet* permite alcance global de recursos, a linguagem de programação Java*[OMG 97]* quebrou os limites impostos pela heterogenidade dos sistemas conectados à rede, o paradigma de reflexão computacional utiliza um modelo de programação em que um sistema pode analisar seu próprio comportamento e atuar sobre ele próprio. Um padrão para acessar objetos distribuídos foi estabelecido por CORBA *[OMG 96]* e implementado por recursos de acesso a *WEB* (*OrbixWeb, IIOP*), permitindo transparência de acesso e localização nas chamadas remotas a métodos de objetos distribuídos na rede.

Por outro lado, as empresas de telecomunicações, oferecem serviços aos assinantes, utilizando tecnologia com sistemas analógicos, pois, a bem pouco tempo, os serviços digitais foram disponibilizados para este setor. Além de utilizarem equipamentos complexos e de fabricação proprietária, e muitos dos serviços oferecidos, ainda são semi-automatizados.

Nos dias atuais, as empresas de telecomunicações, já despertaram para o grande problema da globalização e das privatizações, que impõe que as mesmas, se tornem mais competitivas e competentes para conquistar o maior número de clientes com seus serviços e produtos. Com este novo paradigma de marcado , a evolução e o baixo custo das **Tecnologias de Informação**, as empresas encontraram um novo caminho que abrange em grande parte este novo paradigma de negócios: **a Internet e a Extranet.**

A meta inicial deste trabalho, consiste em simular serviços oferecidos por empresa de telecomunicações na rede *Internet,* gerando bases de informações gerenciáveis (MIBs) que disponibilizará as informações para que o sistema de gerência de redes implemente as

funcionalidades de gerência OSI e também permita que através destas bases, disponibilize as informações para diversos segmentos administrativos da empresa, consultarem, gerarem gráficos, estatísticas, fluxo de trabalho para tomada de decisões e utilizações variadas destas informações através de ferramentas computacionais disponíveis. Este trabalho é uma experiência na implementação de serviços de Telecomunicações utilizando CORBA, Java e Reflexão Computacional [MAES 87], [MAES 88], [FERBER 89], [LISBOA 97].

2. Implementação do Modelo de Gerência de Serviços de Telecomunicações

Esta seção apresenta a implementação de um modelo para gerência de serviços de telecomunicações apresentado em [RICCCIONI 98a, RICCCIONI 98b]. A técnica utilizada para a implementação foi inspirada nos trabalhos de [ORFALI 96] e [ORFALI 97]. A base deste trabalho com relação a distribuição de serviços de telecomunicações está em [PENNA 96].

***Por que JAVA, CORBA e Reflexão Computacional* ?**

2.1 As principais razões para utilização de JAVA são:

- **Portabilidade de aplicações independente de plataforma**: as aplicações JAVA são altamente portáveis, devido a representação padronizada de byte-code, geradas pelos compiladores JAVA, independente de máquina e de sistemas operacionais. Isto significa, um avanço sobre as outras linguagens de programação, em particular para aplicações cliente, uma vez que um único conjunto de código fonte ou de *byte-code* compilado, será utilizado em qualquer plataforma. Consequentemente, o custo de manutenção e desenvolvimento pode ser significativamente reduzido.
- **Programação na *Internet***: as ligações da Linguagem JAVA permite implementação dos clientes CORBA como *applets*. Isto habilita acesso a objetos CORBA, e potencialidade para herdar aplicações encapsuladas em objetos, utilizando os *Browsers* populares. Na realidade os *Browsers* que oferecem o JAVA estão tornando-se uma *Universal Graphical User Interface* (GUI). Em uma empresa a mesma tecnologia pode ser utilizada em *Intranets* , uma vez que o protocolo TCP/IP é utilizado. Embora seja útil possuir *applets* que sejam somente clientes para objetos CORBA, *applets* também podem implementar objetos CORBA. Esta abordagem é limitada devido ao modelo de *applet sandbox*, o qual desabilita o acesso das *applets* a recursos da máquina onde elas são executadas, contudo, as *applets* podem fornecer interfaces **CallBack** (uma chamada de um servidor para um cliente, invertendo a regra Cliente/Servidor).
- **Linguagem de programação orientada a objeto**: Java oferece as mesmas facilidades que as principais linguagens C, C++, SmallTalk, oferecem. Java oferece construções que as linguagens C e C++ não oferecem, ou seja coletor de lixo automático, manipulação de exceção e suporte a *Threads.* Estas características são muito usadas na programação de sistemas distribuídos.

2.2. As principais razões para utilização do CORBA são:

- **Definir interfaces independente de implementação:** A OMG IDL oferece um meio de separar a interface, da implementação dos objetos. Esta separação é muito usada em Engenharia de Software. Sistemas baseados em metodologia e ferramentas Orientadas a Objetos, bem como Técnicas de Modelagem de Objetos (OMT), e a Linguagem de Modelagem Unificada (UML), podem ser escrita com OMG IDL. Uma vez que interfaces são especificadas em IDL, um desenvolvedor ou um grupo de desenvolvedores pode implementar partes diferente do sistema. A separação da interface da implementação também é bastante usada para gerenciar componentes de software. A IDL permite acesso a múltiplas implementações, podendo ainda ser estendida por herança, onde são substituídas por interfaces-base.
- **Independência de linguagem de Programação:** O CORBA fez o mapeamento da IDL para várias linguagens, de maneira que parte do sistema ou de aplicações podem ser

implementadas em diferentes linguagens desde de que a mesma esteja mapeada, pois, toda interação em uma aplicação acorre através da interface que é especificada independente da linguagem de programação que a implementará. Anteriormente, aplicações distribuídas eram implementadas em uma linguagem particular por causa da disponibilização de bibliotecas de acesso remoto da linguagem. Com CORBA escolhe-se a linguagem mais apropriada para implementar o objeto, baseando-se na integração, na experiência e no domínio da equipe de desenvolvedor com a linguagem, ou a linguagem conveniente para implementar a semântica do objeto.

- **Acesso a Objetos independente de sua localização:** Aplicações distribuídas são baseadas em URL ou *Socket*, pois, precisam endereçar o Servidor especificando um *host name* e o número de uma porta de comunicação. Em contraste, CORBA disponibiliza localização transparente, dos métodos independentes do local físico a onde estejam, podendo ser trocado de local sem interromper a aplicação que o utiliza. CORBA ainda disponibiliza mecanismo para recomeçar o serviço solicitado pelo Cliente. Isto pode ser controlado pelo policiamento de vários sistemas servidores.
- **Geração automática de código tratando invocações remotas:** Sistemas distribuídos exigem um esforço de programação de baixo nível incluindo abertura, controle e fechamento de conexão em rede; *marshaling e unmarshaling* ; e um esforço para estabelecer escuta de requisições pela portas de *sockets*, além de solicitar a implementação do objeto. Compiladores IDL e sistemas que utilizam o ORB, deixam o programador livre destas tarefas. Eles criam representações de definições elaboradas em IDL, tais como, constantes, *data types*, e interfaces em uma particular linguagem de ligação, por exemplo, C++ ou JAVA. Também criando o código para o *marshaling e unmarshaling* para os tipos de dados definidos pelo desenvolvedor. Bibliotecas são disponibilizadas para suportarem os tipos predefinidos pelo CORBA.

  O código gerado para o cliente, isto é, o código que invoca uma operação sobre um objeto, é conhecido como *stub*. O código gerado para o servidor, o qual invoca os métodos da implementação que se quer operar, é chamado de código *skeleton.* O código *skeleton* em conjunção com o ORB fornece um mecanismo transparente em tempo de execução do aplicativo, manuseando as solicitações que chegam e administrando as conexões de rede associadas aos objetos.
- **Acesso aos serviços e facilidades do modelo CORBA:** O ORB oferece um meio para chamada distribuída e transparente a métodos de objetos remotos. Normalmente, aplicações distribuídas complexas, necessitam funcionalidades adicionais. Dentro da OMG estas necessidades tem sido analisadas e conduzidas para a especificação de serviços fundamentais. Estes serviços são publicados com o produto *CORBAService*. Por exemplo:
- ***Naming Service* -** serviços nomeação ou referência para objetos distribuídos;
- ***Trading Service* -** serviços negociação para objetos distribuídos;
- ***Event Service* –** serviços assíncronos, de mensagem baseados em assinatura;
- ***Transaction Service* –** processamento de transações para objetos distribuídos;
- ***Security Service* -** disponibilização de autenticação, autorização, criptografia e outras características de segurança.

2.3. A principal razão para utilização de Relexão Computacional :

Este paradigma facilita a replicação e o controle dos objetos utilizados pelos múltiplos clientes dos serviços oferecidos em uma plataforma CORBA. Cada serviço solicitado será mapeado sob a forma de um objeto-base. Os protocolos de coordenação são implementados na forma de meta-objetos. Estas entidades na verdade, nada mais são do que o conjunto de *stubs* no cliente e no servidor, *stubs* para comunicação dos objetos e o BOA (Basic Object Adapter) com suporte para o gerenciamento do grupo de objetos (bibliotecas de objetos, repositório de implementações), gerados todos a partir da tradução da especificação IDL da interface de um objeto-servidor. O servidor é implementado através de um conjunto de métodos da classe servidor, gerada na compilação da IDL, ficando o programador responsável pela inserção dos códigos de cada método da classe. No processo de compilação das interfaces, a plataforma CORBA gera automaticamente todo o suporte para a comunicação (*stubs*) entre as entidades envolvidas, incluindo também as funcionalidades para gerenciamento de grupos de

objetos. O compilador também gera o arquivo contendo o "esqueleto" do código do servidor (declaração de variáveis e métodos). O programador então, fica responsável pela descrição dos objetos-base da aplicação e dos protocolos de coordenação, preenchendo os corpos dos métodos definidos nas interfaces dos meta-controladores. Com este esquema de implementação, os objetos-base cliente e servidor permanecem isentos de quaisquer atividades que não estejam ligadas à aplicação propriamente dita. Todos os aspectos relativos ao gerenciamento do objeto e às interações no contexto CORBA ficam concentrados ao nível de meta-controladores. Neste contexto, o cliente apresenta-se estruturado como um cliente-base que representa o comportamento da aplicação, e um meta-cliente, que não possui função ativa, mas que pode ser usado no gerenciamento de um determinado número arbitrário de clientes, ou para implementar mecanismos de tratamento de exceções no cliente. A estrutura do servidor é semelhante a do cliente, sendo que o serviço, ou seja o objeto que se deseja replicar, é um meta-controlador ou meta-servidor, responsável pelo protocolo de coordenação [VOGEL 97].
Como exemplo, veja a Figura 1, que implementa uma parte do gerenciamento de serviços, o qual controla todos os clientes e suas solicitações e também mostra, através de referência (CORBA *objectreference*), que um cliente pode passar a ser servidor, dependendo da necessidade, fazendo com que o cliente haja sobre ele mesmo. Neste contexto o **Client** apresentar-se-á estruturado em um cliente-base que representa o comportamento da aplicação, e um meta-cliente, que não possui função ativa, mas que pode ser usado no gerenciamento de um número determinado de clientes e o objeto **MultiCoordinator** será um meta-controlador (meta-servidor), responsável pelo protocolo de coordenação. O código a seguir está escrito em IDL e retrata o modulo *Client* e *MultiCoordinator*, mostrando suas interfaces com suas respectivas operações.

```
module Client
{ interface ClientControl
  { // start and stop operations for controlling client counting
    boolean start();
    string  stop();
  };
};
```

```
module MultiCoordinator
{ interface Coordinator
  { // object for registering and controlling client
    boolean register(in string clientNumber,
          in Client::ClientControl clientObjRef);
    boolean start();
    string  stop();
   };
};
```

2.4. Implementação dos módulos

Os serviços serão objetos CORBA, ou seja, terão interface definida em IDL de modo a serem acessíveis por qualquer objeto e também acessam qualquer outro objeto CORBA, independentemente da linguagem de programação. Os serviços serão identificados por sua referência a objetos e assim localizados pelo *brokers*. Ver Figura 1.
Nesta etapa, foram analisados alguns requisitos para implementação de sistemas na WEB:
Os **objetos gerenciáveis** foram modelados utilizando-se técnicas de orientação a objetos. Quanto a gerência destes objetos, por serem serviços processados em tempo real, ou seja, OLTP (*OnLine Transaction Processing*), executam simultaneamente e repetidamente um conjunto predefinido de declarações SQL (*Structured Query Language*).
Os clientes devem requisitar um serviço, fazer suas coisas, e então partirem. A função básica da OLTP é a reutilização de recursos. Não é permitido a ninguém monopolizar recursos escassos. Todo mundo deve aprender como compartilhar.
Nas últimas três décadas, os projetistas de OLTP dominaram a arte de comprimir ao máximo a performance fora de seus sistemas. Eles sabiam como escrever aplicações que escalonavam. A medida importante para os projetistas de aplicações OLTP eram *thoughput*, tempo de resposta , e o número de usuários concorrentes.
A seguir está a lista do que possibilitamos medir na aplicação de Gerência de Serviços de Telecomunicações:

- ***Benchmark* dos serviços,** o *Benchmark* dos serviços é um modelo hipotético para uma aplicação de serviços de telecomunicações oferecidos na WEB. Ele define o padrão de transação chamado *User*, uma base de dados padrão chamada **Mib_Servico**, um método

escalonável para grandes sistemas, e uma medida de *throughput* e de preço por performance;

- **Atualizações em tabelas** da base de dados, para melhorar o atendimento e a *performance* do servidor de serviços;
- **Relatório de Eventos** para tomada de decisões, caso os eventos sejam críticos e para ativar técnicos no sentido de realizarem manutenções corretivas ou preventivas a equipamentos que apresentam falhas (a ativação de técnicos para manutenção não foi implementada).

Algumas definições são destacadas:

1. *Throughput*: esta é a taxa na qual o sistema OLTP pode processar transações. Medimos o *throughput* em transações por segundo (tps). A medida que adicionamos usuários, o sistema alcança o máximo *throughput* e então começa a cair, a medida que mais usuários entrarem no sistema. Um bom sistema OLTP deve ser capaz de manter o *throughput* estável, quando adicionarmos mais usuários.

2. *Response Time*: este é o tempo transcorrido desde o momento em que você submete a transação até o momento em que você receber a resposta. O tempo de resposta deve crescer linearmente, não exponencialmente, a medida que acrescentamos mais usuários ao sistema.

3. *Concurrent users*: o número de usuários concorrentes é um fator importante nos sistemas OLTP. *Throughput* e tempo de resposta são significativos quando os medimos em função do número de usuários. Devemos também medir a *performance* do sistema em momentos de pico. O gerente certamente estará interessado em números de *performance* de pico .

4. A definição de transação: para a unidade tps ser significativo é importante que concordemos o que exatamente é uma transação. Uma transação varia de aplicação para aplicação. Uma transação que reserva um lugar numa linha aérea é obviamente diferente de uma transação de chão de fábrica. Nosso *beachmark* deve ser capaz de especificar exatamente o que é uma transação. Por exemplo, devemos descrever ambos, os comandos reais SQL e também como aplicamos, *commits*, *logs*, e *locks*.

5. *System steady_state verification* (Verificação com o sistema estabilizado): Um sistema alcança o *steady state* quando sua *performance* fica constante de um determinado ponto em diante. A *performance* do sistema pode variar 20% ou mais antes de atingir o *steady_state*. Podemos atribuir esta variação a diversos fatores, incluindo *buffers* que enchem e ao *swapping* de páginas da memória para disco, a medida que a carga do sistema aumenta. Devemos relatar os resultados dos *beachmarks* quando o sistema estiver nas condições de *steady_state* na tabela de eventos.

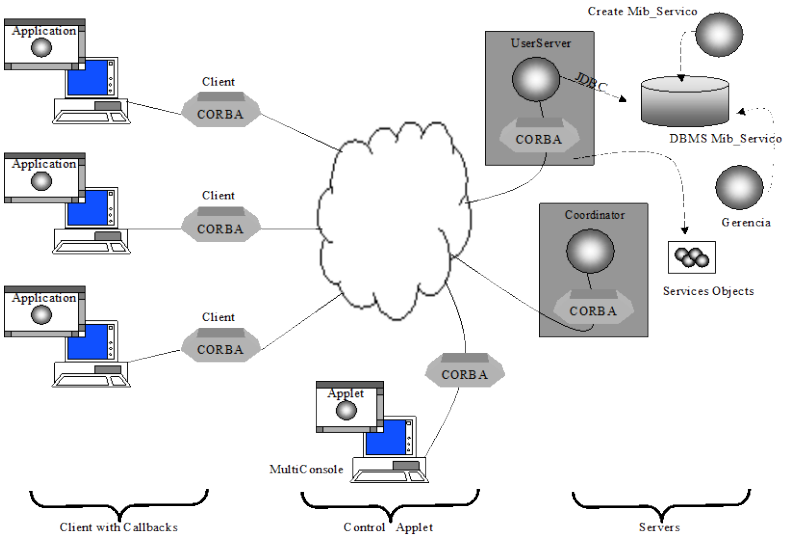
6. *Cost per transaction* (custo por transação): o custo do sistema deve ser usado para normalizar resultados de testes de tps através de sistemas amplamente diferentes, por exemplo, um PC de $3000 versos um *Mainframe*. Isto significa que devemos mensurar o custo por *tps*.

7. The test environment(o ambiente de teste): Nosso *benchmark* deve especificar o *hardware* do sistema e as configurações de *softwares*. A configuração de *hardware* deve incluir a velocidade do processador, o montante de memória, o tamanho do *buffer pool*, e a velocidade dos discos. Devemos também especificar o ambiente de comunicação entre o cliente e o servidor.

8. *Full disclosure* (declaração ampla): Devemos fornecer informações suficientes para permitir a um auditor, replicar os resultados dos nossos testes.

Para alcançar estes objetivos de gerência, utilizei como base a Figura 1 e estruturei o trabalho da seguinte forma:

- ✓ Descrição da IDL – para declaração formal do modelo de gerência;
- ✓ Criação de uma base de dados gerenciável, com base nos atributos definidos no modelo de gerência;
- ✓ Criação de um ambiente computacional compatível com as necessidades do modelo de gerência;
- ✓ Criação de páginas em HTML, para escolha e solicitação do serviço a ser executado;
- ✓ Implementação dos objetos (Cliente e Servidor) .




```
module service
{
 exception BankException
  { string reason;
  };
interface User
 {
  string          receiveservice(in string usr, in string prog);
  string          Consultausr(in string usr, in string pwd);
  void            AddCentralComut();
  string          consContChamada (in string usr, in string prog);
  void            decCentralComut();
  void            addServico (in string usr, in string prog);
  void            UpServico (in string usr, in string prog);
};
// Distribuidor de objetos (serviços solicitados)
 interface Dispenser
 {
    User        reserveUserObject() raises (BankException);
    void        releaseUserObject(in User userObject)
                      raises (BankException);
 };
};
```

```
// Coordinator.idl
module Client
{ interface ClientControl
  { // start and stop operations for controlling client
counting
    boolean start();
    string  stop();
  };
};
module MultiCoordinator
{ interface Coordinator
  { // object for registering and controlling client counting
    boolean register(in string clientNumber,
                  in Client::ClientControl clientObjRef);
    boolean start();
    string  stop();
  };
};
```

Figura 1. Cenário da aplicação de Gerenciamento de Serviços com sua IDL.

Os atributos dos objetos na MIB, foram implementados em um Sistema Gerenciador de Banco de Dados Relacional, o qual dispõe suas informações para qualquer aplicação que necessite dessas. Este SGDB, deve ter todas as características de um banco de dados de "primeira linha" (ORACLE, SYBASE, SQL-SERVER,DB2,...), e que suporte aplicações Cliente/Servidor. Como nosso exemplo é meramente acadêmico usamos o *MS-ACCESS-97 da Microsoft*, por ser um banco de dados que atende as necessidades deste trabalho, porém não suporta grandes capacidades de informações. O modelo desta MIB está descrito na Figura 2.

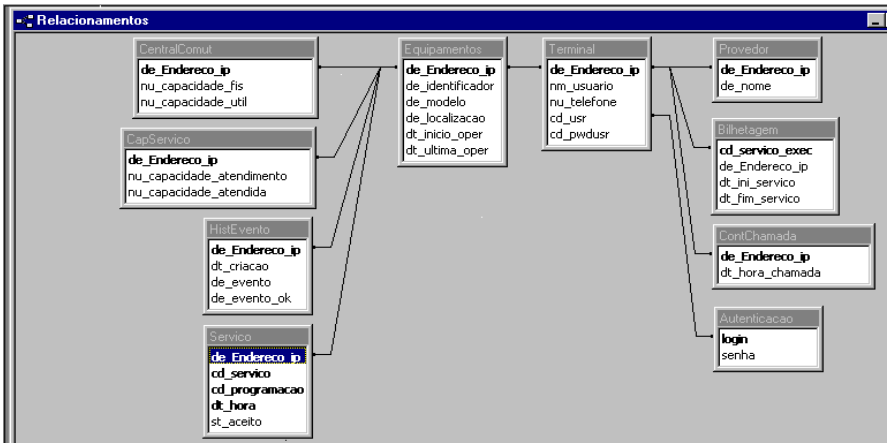


Figura 2. Base de Informações de Serviços – MIB-Serviço

Com a utilização da Internet para o gerenciamento, temos que dar a devida atenção a questão de segurança, com a utilização de canais seguros como HTTPS e SSL para a autenticação, bem como na manutenção das informações, pois nem todos os *Browsers* estão aptos com o protocolo IIOP, que viabiliza IOR através do *Gatekeeper* (Figura 3), o qual funcionará como um *firewall*.

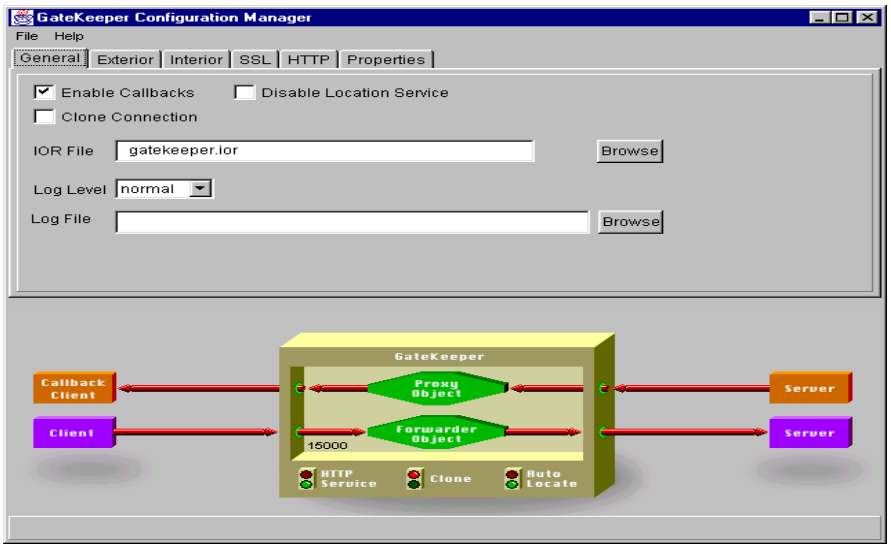
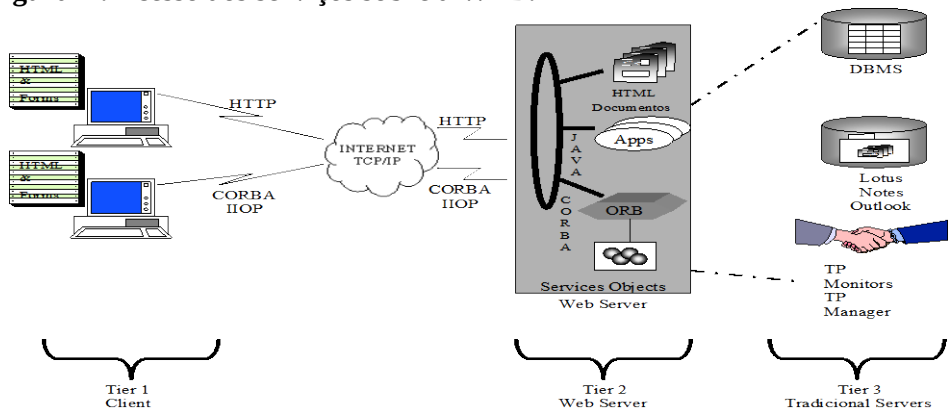


Figura 3. Ferramenta GateKeeper, controle de segurança.

Na Figura 4, o acesso às bases de dados remotas são feitas através de aplicações CORBA, dando mais segurança às transações remotas, com o auxílio de seus serviços e protocolo (CORBA*services* e CORBA IIOP). Isto causa um *overhead* menor, embora perca *performance*, pois, se utilizássemos para implementação um acesso direto às bases de dados com JDBC, seria bem mais rápido o acesso aos dados, porém, teríamos que implementar vários controles na aplicação-cliente e no ambiente Cliente/Servidor, como por exemplo: *firewall*, *HTTP-S*, *SSL* e outros. Isto não é uma boa solução para sistemas distribuídos na WEB. Por este motivo, foi criado um servidor WEB, com o Windows-NT *Server*, utilizando a ferramenta *Internet Information Server* que possibilita a criação de um *Site*, permitindo acesso as página HTML, *applets* e outros componentes da aplicação proposta, a partir do servidor WEB.

Figura 4. Acesso aos serviços sobre a WEB.



O coordenador de objetos (CoordinatorImpl) será um meta-objeto que contém informações sobre todos os outros objetos, o qual trata qualquer número de serviços, de qualquer tipo, em *threads* independentes, e com igual prioridade. Este meta-objeto também é conhecido como Reflexão Computacional, que tem como função básica explicitar como o objeto reage diante de uma mensagem, possibilitando a intervenção no estado de sua computação. Em nossa implementação usamos a linguagem Java para implementar o coordenador com *threads* e o *Callback* (Figura 5), o tornando Cliente e/ou Servidor, dependendo do estado da computação.

O Coordenador (Figura 5) permitirá capturar o estado dos serviços ativando e desativando-os, através de comandos externos ao mesmo.

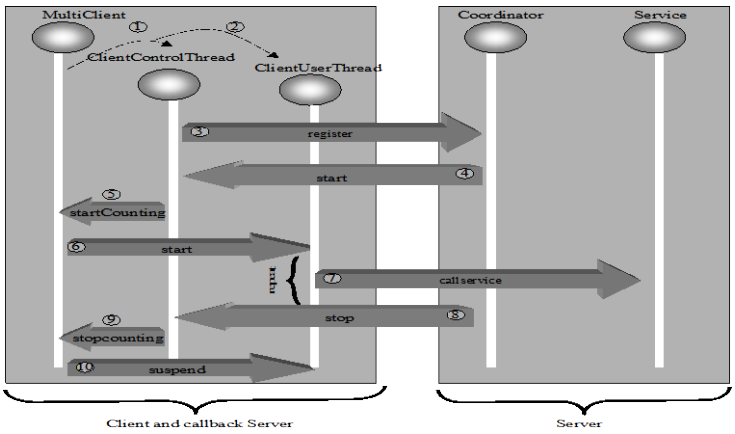


Figura 5. Cenário de execução interna do objeto cliente com Callback.

A **Gerencia e o MultiConsole -** São aplicações de Gerência de Serviços, são executada de qualquer ponto da WEB, permitem ao gerente analisar e manter o bom funcionamento do servidor e dos serviços solicitados pelos clientes. O cenário das classes acima descrita pode ser melhor compreendido observando a Figura 6.

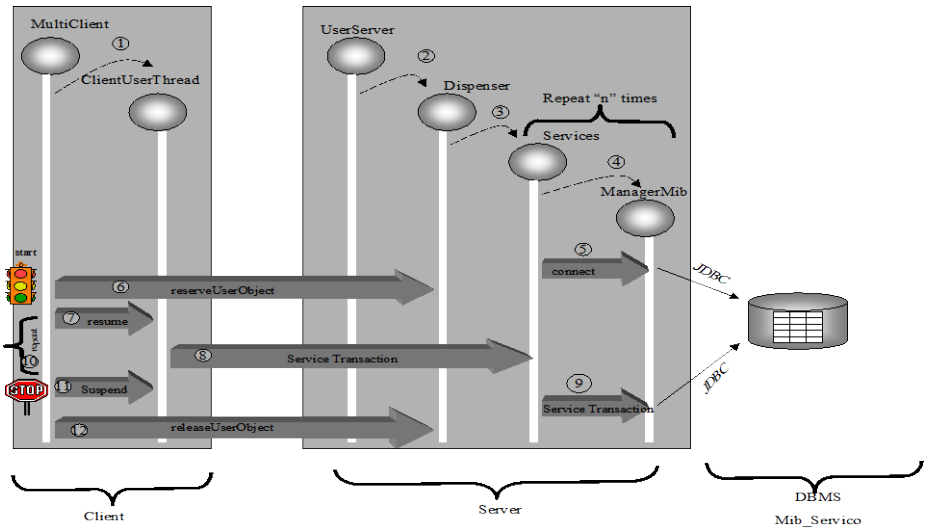


Figura 6. Cenário de execução interna do serviço.

3. CONCLUSÕES

O modelo implementado neste trabalho, para **Gerenciamento de Serviços de Telecomunicações**, consiste em focalizar alguns pontos relevantes às aplicações distribuídas, em combinação com ambientes heterogêneos e algumas vezes proprietários e que com isso promovam subsídios suficientes e necessários para a avaliação supra citada.

Não foi objetivo deste trabalho simular fielmente as características, o comportamento e a gerência dos equipamentos reais do ambiente de telecomunicações que prestem serviços aos usuários e tampouco prover tais serviços. Como a proposta da aplicação é apenas uma simulação simples e abstraída sem compromisso com o mundo real, e também objetivando uma clareza acadêmica, a aplicação constitui-se de dois subsistemas: um gerenciado, o qual simula o serviço citado; e outro gerenciador, o qual exerce algumas atividades de gerência sobre o primeiro.
Como considerado no escopo deste trabalho, a aplicação-gerente procurou tanger os aspectos clássicos da distribuição, que possibilitaram o estudo e a experiência com plataformas de distribuição, no caso o CORBA. Na comunicação entre aplicação gerente e a aplicação gerenciada foram usados recursos do ambiente Java. Tal experiência permitiu a verificação das possibilidades, das facilidades e das dificuldades da interação entre CORBA e o Java. Para se alcançar as metas foi simulado o serviço despertador automático e uma aplicação de gerência para este serviço. Esta aplicação de gerência exerce atividades simples de gerência sobre a simulação do serviço de despertador, utilizando características reflexivas. Ao se separar os aspectos de gerência, da própria construção das aplicações de serviços, usou-se a técnica de **Reflexão Computacional,** além de oferecer condições para o uso e o teste das principais características da arquitetura CORBA. Esta arquitetura usa o mecanismo do ORB, para prover o meio de transporte das informações. Desta forma conseguimos unir as vantagens oferecidas por este novo paradigma aos benefícios dos serviços definidos como objetos CORBA. Isto possibilita que outros objetos CORBA, localizem um ao outro, independente de localização e da linguagem de programação usada, além de usufruir da interoperabilidade entre meios heterogêneos.
O uso do Visigenic 3.x, ofereceu a facilidade do mecanismo de *callback* e a linguagem de programação Java, a disponibilidade de várias estruturas prontas e que atendiam necessidades específicas como as classes *vector, threads* e o método *constructor*.
Os serviços de telecomunicações não se constituem em um paradigma, relacionando-se apenas às aplicações na Internet, como uma nova proposta de oferecê-los aos clientes.
Em sistemas distribuídos de modo geral, a possibilidade de transferir a computação para outro ponto da rede pode propiciar o desenvolvimento de soluções mais eficiente para certos problemas como por exemplo, gerência de rede, gerência de sistemas, automação de *WorkFlow*, gerência de documentação eletrônica, *Benchmark* de aplicações, gerência de Banco de Dados e outros.
O atual enfoque deste trabalho, buscou satisfazer as diferentes tecnologias existentes (analógica e digital) e torná-las interoperavéis, fugindo do escopo onde se buscava definir padrões, o que causou o

aparecimento de tecnologias proprietárias no setor de telecomunicações, bem como em outros setores que utilizam a tecnologia de rede como ferramenta imprescindível ao desenvolvimento de aplicações distribuídas.
Ao elevarmos o nível de abstração, a proposta deixou em aberto o meio de se implementar os serviços, permitindo assim o uso de qualquer mecanismo independente de CORBA-Java, o que na prática poderá não ser eficiente. A interoperabilidade entre sistemas na WEB, já está comprometida pela dependência da linguagem de programação.
A aplicação de *Benchmark*, usada neste modelo de Gerência de Serviços de Telecomunicações, poderá ser utilizada em qualquer sistema que se preocupe com gerência de recursos computacionais. Da maneira que esta aplicação foi feita, o responsável pela gerência, é capaz de avaliar: se a configuração do ambiente computacional está em conformidade com o seu planejamento ou se precisa ser alterada; se o custo/performace de todo o sistema de gerenciamento de serviço está adequado aos gastos previstos.
Esta aplicação de *Benchmark* foi desenvolvida para WEB, podendo ser ativada a qualquer momento pelo gerente, e de qualquer lugar. Esta aplicação foi escrita em linguagem Java, e com referências a objetos, controlados pelo controlador de objetos da aplicação, permitindo com isso, as avaliações mencionadas acima.
Os acessos à Base de Dados Remotas, poderiam ser mais rápidos, caso fizéssemos, os acessos diretamente ao SGBD via JDBC, porém, por questão de segurança, os acessos foram feitos através do ORB a um servidor remoto, que têm implementado todos os acessos à Base de Dados do sistema de gerenciamento de serviços. Outra vantagem, além da segurança, é a reutilização e a padronização dos códigos de acesso à Base de Dados, tornando as manutenções futuras, facilmente realizadas, bem como, novas implementações de aplicações de um modo geral.
A utilização de *object reference*, foi um poderoso recurso oferecido pelo CORBA, na negociação de serviços distribuídos, bem como o *callback* que é muito eficiente para o controle de clientes em aplicações Cliente/Servidor.
Esse trabalho detectou, que aplicações na WEB, que manipulam hora, devem ter uma atenção especial quanto a sincronização de relógio, tanto por parte do Cliente, quanto por parte do Servidor. Além desta sincronização, temos que dar a devida atenção a horários de verão regionais, pois, prejudicam sensivelmente estas aplicações.
Outro aspecto relevante neste trabalho é o fato de que a maioria das aplicações WEB baseadas em dados, sofrem com a falta de velocidade ou largura de banda dos meios de comunicação. Como a natureza da WEB é *stateless*, ou seja, sem estado e também, sem conexão, cada vez que um Cliente requisita dados ao Servidor, esta requisição precisa conter todas as informações de estado que são necessárias para satisfazer a solicitação. Portanto, o Servidor, tipicamente, executa uma nova consulta à base de dados, ao invés de reutilizar os conjuntos de resultados previamente estabelecidos. O Servidor gera também um novo documento HTML a cada vez, contendo os dados selecionados e pré-formatados para exibição pelo *browser* WEB. Com o ambiente CORBA, melhorou-se o desempenho e o tempo de resposta ao Cliente, e ao mesmo tempo, reduzindo o número de requisições que o Cliente faz ao Servidor e a quantidade de trabalho que o Servidor precisa executar. Para se conseguir este melhor desempenho, documentos HTML comuns ou com *applets*, descrevem como os dados devem ser exibidos, porém os dados em si são fornecidos ao Cliente separadamente, através do *stub* CORBA da aplicação. Essa divisão de responsabilidade permite que o Cliente visualize, pagine, busque e ordene os dados, localmente, atualizando a tela conforme necessário sem fazer novas conexões ao Servidor.
Concluindo, podemos afirmar que este modelo de gerência é flexível à medida que pode ser implementado por um outro ORB, o qual possua mapeamento IDL-Java. Esta exigência não chega a ser comprometedora, visto que, entre as linguagens mais adequadas para a implementação de aplicações na WEB, a linguagem Java é a mais eficiente.

4. REFERÊNCIAS BIBLIOGRÁFICAS

[FERBER 89] Ferber J., Computational Reflection in Class Based Object-Oriented Language. SIGPLAN Notices New York, v.24, n.10, pag.317-326, october 1989.

[LISBOA 97] **Lisboa Black, Maria L.,** Reflexão Computacional no Modelo de Objetos Texto apresentado na Universidade Federal do Rio Grande do Sul CPGCC, UFRGS, 1997. Tese de Doutorado

[MAES 87] Maes P., Concepts and experiments in computational reflection. SIGPLAN Notices, New York, v.22, n.12, p. 147-169. Trabalho apresentado no OOPSLA, 1987, Orlando, Flórida.

[MAES 88] Maes P. ,Issues in computationl reflection. Meta-Level Architecture and Reflection. Amsterdan Elsevier Science, 1988.

[OMG 96] Object Management Group, The Common Object Request Broker 2.0/IIOP specification Revision 2.0, OMG Document , agosto 1996.

[OMG 97] Object Management Group, Java Language Mapping RFP OMG Document , março 1997.

[ORFALI 96] **R.Orlafi, D. Harkey, J. Edwards,** The Essencial Distributed Objects - Survival Guide

John Wiley & Sons, Inc, 1996

[ORFALI 97] **R.Orlafi, D. Harkey,** Client/Server Programming with JAVA and CORBA

John Wiley & Sons, Inc, 1997

[PENNA 96] **Manoel Camillo Penna, Andrey Patiticci,** Relatório Técnico à respeito de Sistema de Gerenciamento Distribuído, com exemplo sobre despertador automático.

[RICCIONI 98a] Paulo Roberto Riccioni Gonçalves, Gerenciamento de Serviços de Telecomunicações com CORBA e Java, dissertação de mestrado, CPGCC/UFSC, agosto de 1998.

[RICCIONI 98b] Paulo Roberto Riccioni Gonçalves e João Bosco Mangueira Sobral , Telecommunication Services Management with Computational Reflection, using CORBA and Java, ITS`98 Proceedings SBT/IEEE International Telecommunications Symposium, São Paulo, Agosto 1998.

[VOGEL 97] Andreas Vogel & Keith Duddy , Java Programming with CORBA John Wiley & Sons, Inc. - 1997.