Estratégias para a Integração de Aspectos de Tempo Real em uma Aplicação Distribuída Legada

Juliana R. B. D. Barros, Carlos A. G. Ferraz
Universidade Federal de Pernambuco, Centro de Informática,
Caixa Postal 7851, 50732-970, Recife-PE, Brasil
{jrbd,cagf}@cin.ufpe.br

Resumo

Aplicações distribuídas com suporte a tempo real fazem parte de um tipo emergente de aplicação. Neste tipo de sistema podem ser encontrados alguns fortes requisitos de qualidade de serviço e recursos temporais. Embora exista esta tendência de utilização de sistemas que atendam a requisitos de tempo e qualidade de serviços, os sistemas que já atingiram uma certa maturidade em ambientes corporativos não podem simplesmente ser abandonados e re-desenvolvidos com a intensão de atender a tais restrições. Com o objetivo de encontrar uma solução para isto, este artigo apresenta duas estratégias de migração de sistemas distribuídos legados para sistemas de tempo real, utilizando as especificações de Real-time CORBA e CORBA Messaging.

Palavras-chave: Sistemas Distribuídos, Real-time CORBA, Sistema Legado, Estratégias de Integração.

Abstract

Distributed applications are steadily growing in diverse areas and gaining space in systems where requirements of quality of service and time response are a determinant factor. Although there is a trend of using systems that attend to time and quality of service requirements, systems that have already reached certain maturity in enterprise environments can not simply be abandoned and be developed again with the purpose to fulfill such restrictions. In order to find out a solution to this, this work presents two strategies to migrate distributed legacy systems to real-time ones, making use of specifications of Real-time CORBA and CORBA Messaging that aim at supporting those requirements stemmed from real-time distributed systems.

Keywords: Distributed Systems, Real-time CORBA, Legacy System, Integration Strategies

1 Introdução

Com o advento da Internet, o aumento da interatividade e o progresso da engenharia de software, os usuários tornaram-se mais exigentes no que diz respeito à qualidade de serviço, particularmente com relação, ao tempo de resposta. As aplicações distribuídas estão em constante crescimento em diversas áreas de atuação e ganharam espaço em sistemas onde os requisitos de qualidade de serviço dependentes de tempo são considerados fatores determinantes.

Sistemas que devem reagir ao estímulo de um objeto controlado ou do operador, dentro do intervalo de tempo especificado pelo seu ambiente são considerados de tempo real. Para que aplicações distribuídas atendam aos requisitos de tempo real faz-se necessária a utilização de plataformas de suporte à distribuição que implementem características de tempo real.

A primeira geração de ferramentas de suporte à distribuição não apresentava esta característica, pois seus fabricantes não tinham em mente este objetivo, sobretudo porque o enfoque principal estava em facilitar o desenvolvimento de aplicações distribuídas.

Embora exista esta tendência de utilização de sistemas que atendam a requisitos de tempo e qualidade de serviço, os sistemas que já atingiram certa maturidade em ambientes corporativos não podem ser simplesmente abandonados e re-desenvolvidos contemplando tais restrições, pois os mesmos continuam em operação e as empresas, em geral, não apresentam recursos para re-

desenvolvimento de praticamente todos os seus sistemas de forma a se adequar a esta nova realidade.

Um sistema computacional legado pode ser definido como aquele que foi projetado e implantado há um certo tempo, de modo que a sua utilização já foi consolidada [1]. Ele já faz parte do dia a dia dos usuários e qualquer alteração a ser realizada sobre ele deve ser feita de forma gradual, sem que prejudique a sua utilização.

Com o objetivo de encontrar uma solução para a situação de impossibilidade de re-desenvolvimento, este artigo apresenta duas estratégias de migração de sistemas distribuídos legados para sistemas de tempo real, utilizando uma arquitetura padrão de distribuição que se propõe a atender aos requisitos dos sistemas distribuídos de tempo real. Assim, é moldado um novo sistema, o qual, por sua vez, não é necessário ser re-desenvolvido e deve atender a padrões de qualidade estabelecidos durante o seu projeto de integração, apenas tendo seus objetos críticos migrados para uma plataforma de distribuição com suporte a tempo real. Tais objetos mantêm compatibilidade e integração com os demais objetos do sistema que permanecem na plataforma de distribuição original.

A arquitetura CORBA - *Common Object Request Broker Architecture* [2], cuja especificação foi desenvolvida pela OMG (Object Management Group), permite que clientes invoquem operações em objetos distribuídos sem se preocupar com alguns detalhes como localização do objeto, linguagem de programação, plataforma de sistema operacional, protocolos de comunicação, interconexões e hardware. CORBA é hoje uma das plataformas de distribuição mais usadas para o suporte a aplicações de grande porte [3].

Para que a integração de aspectos de tempo real ao sistema distribuído legado seja eficiente é necessário que aquelas operações consideradas críticas, no que diz respeito ao tempo de resposta, sejam executadas sobre um ORB (Object Request Broker) com suporte a tempo real e devem interoperar com os demais objetos e operações do sistema legado que, por sua vez, foram projetados sobre um ORB sem suporte a tais aspectos. Sendo assim, os componentes com suporte a tempo real implementam algumas estratégias definidas a seguir, utilizando aspectos da especificação de uma arquitetura distribuída que estende a arquitetura CORBA – Real-time CORBA.

Durante a definição das estratégias, alguns trabalhos na área, além das duas especificações produzidas pela OMG - Real-time CORBA [4] e CORBA Messaging [5], puderam ser avaliados. O primeiro trabalho relacionado foi realizado pela Highlander e referenciado em [6]. Este trabalho fez um levantamento de um sistema de tempo real para monitoração e gerenciamento de redes, onde foram identificados dentro deste ambiente quais os *deadlines* que esse sistema deveria atender considerando-o um sistema de tempo real. Foi aplicada a política de *Threadpools* definida pelo Real-time CORBA. Essa política também foi aplicada em uma das estratégias aqui definidas. Outro trabalho bastante atual é o que vem sendo realizado pelo grupo DOC (Distributed Object Computing) da Washington University com a utilização de recursos definidos no Real-time CORBA a um sistema de limpeza de agentes tóxicos de uma superfície planetária [7].

Estes trabalhos apenas utilizam os recursos definidos pelas especificações, enquanto que o trabalho aqui apresentado define estratégias que combinam os recursos propostos pelas especificações. O escopo do presente trabalho é mais abrangente, pois tem o objetivo de definir estratégias de integração de sistemas legados para sistemas de tempo real, considerando qualquer sistema distribuído que utilize a plataforma CORBA, onde é mais viável a sua migração que o seu re-desenvolvimento sobre uma plataforma de tempo real.

A validação das estratégias propostas é viabilizada através de estudo de caso, que demonstra que esta alternativa de integração sem a necessidade de re-desenvolvimento total de um sistema legado, classificada como de menor custo, é possível de ser empregada.

Este artigo está dividido em 5 seções. Inicialmente foi apresentado a introdução, e em seguida faz-se a apresentação das estratégias de integração, bem como os conceitos utilizados em cada uma delas. Na terceira seção é apontado o estudo de caso onde as duas estratégias são aplicadas a um sistema legado de Educação a Distância, que implementa uma ferramenta de co-autoria de aulas [8]. Os resultados obtidos com os testes realizados com esta ferramenta integrada a aspectos de tempo real em uma rede metropolitana de alta velocidade são apresentados na quarta seção e então comparados com a ferramenta original, sem aspectos de tempo real integrados. Por fim, as considerações finais e propostas de trabalhos futuros são apontadas e o artigo é concluído.

2 Estratégias de Tempo Real Definidas para o Projeto

Considerando a proposta de integração de aspectos de tempo real em uma aplicação distribuída legada, são definidas duas estratégias cujo suporte é dado pelas especificações de Real-time CORBA e CORBA Messaging e podem facilmente ser aplicadas a um sistema legado. Em [9] são verificadas que características os ORBs devem atender para que possam suportar um sistema de tempo real. Desta forma, os objetos que necessitam de controle mais estrito das suas operações devem ser migrados para um ORB que suporte tais facilidades. Também em [9], podem ser analisados três ORBs baseados em CORBA que apresentam suporte a tempo real e concluí-se que o ORB TAO [10], da Washington University apresenta-se mais completo, no que diz respeito a especificação do Real-time CORBA. Desta forma, este produto é tomado como base na definição das estratégias.

As técnicas de mapeamento de prioridades definidas no Real-time CORBA fazem com que o projetista de sistemas distribuídos de tempo real programe sua aplicação sem muito interesse em detalhes de baixo nível das prioridades de *threads*. Vale a pena salientar que o mapeamento de prioridades do Real-time CORBA não poderá prover garantia total, caso se utilize um sistema operacional que não seja de tempo real. Quando o sistema operacional utilizado não é de tempo real, a infraestrutura de comunicação não pode se comportar de maneira completamente determinística.

Inicialmente são feitas as considerações para a definição das estratégias. Considera-se que o sistema distribuído legado foi desenvolvido dentro da abordagem de objetos distribuídos, utilizando a plataforma CORBA de distribuição, ou seja, foi projetado sobre um ORB compatível com o padrão CORBA, mas sem suporte ao Real-time CORBA.

A aplicação legada deve ser avaliada com relação aos métodos de seus objetos, observando-se quais deles apresentam necessidade de tempo de resposta estrito no que diz respeito às suas solicitações. Assim, os objetos e os seus respectivos métodos são analisados um a um, e de acordo com a sua atuação dentro do sistema, decide-se se um método em particular possui a necessidade de tempo de resposta estrito ou que seja executado a uma maior prioridade dentro do sistema. Aqueles objetos que apresentam um grande número de operações com controles temporais e de prioridades devem ser migrados ao ambiente de tempo real. Os demais não precisam sofrer alterações no seu funcionamento original.

Desta forma, é projetado um novo sistema, considerado híbrido, apresentando algumas restrições com relação a critérios de prioridades e de tempos de execução. Este novo sistema contém uma integração entre aspectos legados e de tempo real.

Vale salientar que esta integração será válida quando o novo sistema integrado não precisa prover uma grande quantidade de ações em tempo real, pois neste caso, o custo da integração é menor que o re-desenvolvimento total. A opção de re-desenvolvimento se justifica apenas para o caso em que a aplicação apresenta todas ou quase todas as ações como prioritárias ou com fortes restrições de tempo.

A Figura 1 mostra o esquema de uma arquitetura de um sistema distribuído legado passando pela integração de aspectos de tempo real. Este sistema é considerado um sistema de tempo real *soft*, e possui três objetos que devem ser migrados para um ORB com suporte a tempo real e outros três objetos que permanecem sobre a plataforma de distribuição inicialmente utilizada para o sistema legado. A plataforma de distribuição com suporte a tempo real está dividida em duas camadas. A primeira delas refere-se à arquitetura CORBA e sobre a mesma estão as especificações de Real-time CORBA e CORBA Messaging utilizadas nas estratégias aqui definidas. Estas camadas estão localizadas no nível de middleware. Acima deste nível, encontra-se o nível de serviços, no qual estão presentes os serviços de nomes e de eventos, dentre outros serviços CORBA, bem como o elemento RT-Adapter que insere requisições de tempo real, que devem ser atendidas pelos objetos localizados no nível de objetos, e sobre a arquitetura de distribuição que implementa tempo real.

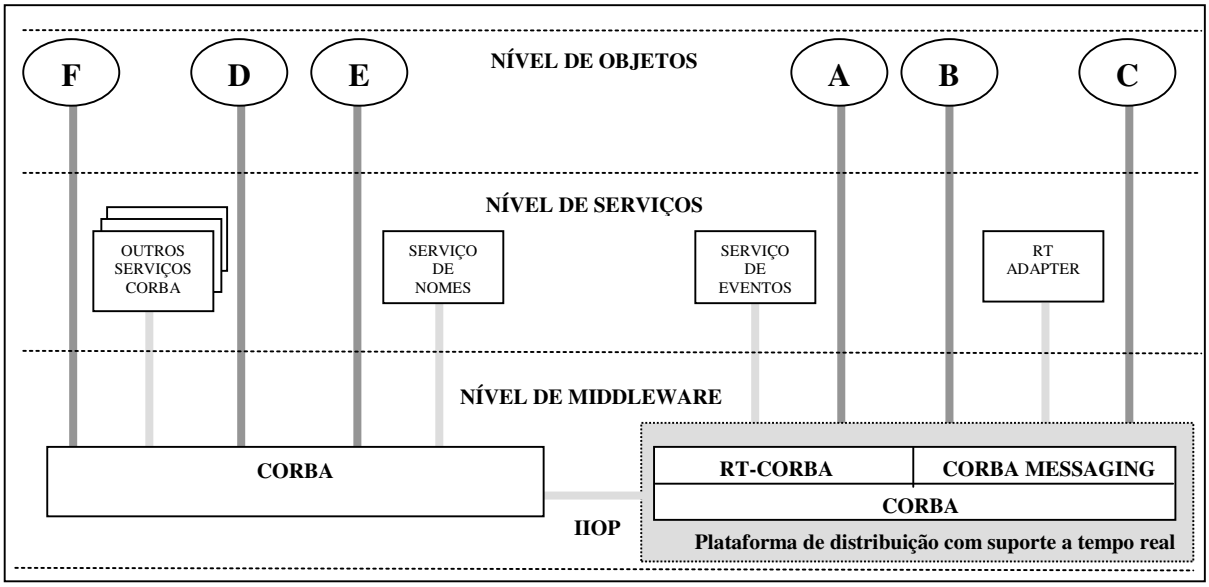

Figura 1 Arquitetura de Integração de Aspectos de Tempo Real ao Sistema Legado

As duas estratégias podem ser resumidas de acordo com a Tabela 1. Nesta tabela é possível identificar que recursos estão sendo empregados em cada uma das estratégias, que são detalhadas nas subseções seguintes.

Estratégia 1		**Estratégia 2**	
RT-CORBA	CORBA MESSAGING	RT-CORBA	CORBA MESSAGING
ThreadPools Modelo SERVER_DECLARED Mapeamento Linear	RelativeRoundTripTimeout	Modelo CLIENT_POPAGATED Mapeamento Linear	RelativeRoundTripTimeout

Tabela 1 Resumo das Estratégias

2.1 Estratégia 1

Para suportar a importância relativa de cada um dos objetos do sistema podem ser configurados diferentes RT-POAs (Real-time Portable Object Adapters) e utilizar o modelo de prioridades declarado no servidor. Esta primeira estratégia aborda o uso de *Threadpools* [11] e o modelo SERVER_DECLARED combinados com a política *RelativeRoundtripTimeout* definida no CORBA Messaging.

2.1.1 Threadpolls

A utilização de *Threadpools* permite o controle dos recursos da *thread* usados pelo servidor em um sistema Real-time CORBA quando processa as solicitações clientes.

Através desta abstração é possível controlar o número de *threads* usadas, alocá-las nas diferentes partes do sistema e obter previsibilidade através do benefício de ter *threads* pré-alocadas quando as invocações são recebidas.

Real-time CORBA define uma API (Application Program Interface) para criação de *threadpools* e a associa com Real-time Portable Object Adapters (RT-POAs). Um número de *threads* são estaticamente pré-alocadas no momento da criação do *Threadpool*. Adicionalmente, um *threadpool* pode ser configurado para permitir que um número de *threads* extras possa ser criado sob demanda quando solicitações são recebidas, mas todas as *thread*s estáticas estão atualmente ocupadas servindo às solicitações previamente recebidas.

Cada RT-POA está associado com um único *threadpool*, mas um único *threadpool* pode ser compartilhado por mais de um RT-POA ou ser exclusivo a um RT-POA.

Os *threadpool*s também suportam o enfileiramento de solicitações. Quando um número máximo de solicitações de *thread*s está em uso, um *threadpool* pode enfileirar as solicitações subseqüentes até o limite configurado e segurá-las até que esteja disponível para processá-las. A vantagem neste enfileiramento é que, uma vez que a mensagem é recebida pela camada de rede, os *timeouts* do protocolo e os re-envios não acontecerão.

Para efeito de simplicidade, considera-se uma aplicação com três objetos com necessidade de suporte a tempo real, de modo que, associa-se o RT-POA com maior nível de prioridade declarado no servidor, para aquele objeto que possuir um maior número de operações a serem executadas a uma alta prioridade. Para aquele objeto, dentre os de tempo real, que apresente o menor número de operações críticas, deve ser associado o RT-POA com menor nível de prioridade. Por fim, dentre os três objetos considerados inicialmente, aquele que apresenta um nível intermediário de operações críticas é associado ao RT-POA com nível intermediário de prioridade. Esta situação pode ser visualizada a partir da Figura 2. Nesta figura, observa-se o uso de *threadpool*s separados com diferentes prioridades para suportar a relativa importância dos objetos da aplicação.

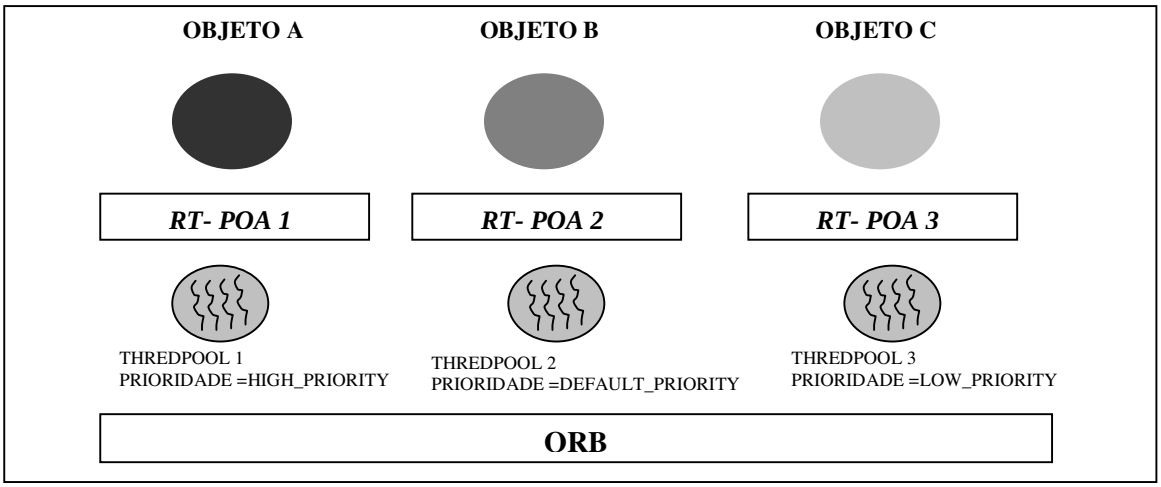

Figura 2 Configuração dos Threadpools

2.1.2 Modelo de Prioridades

Em paralelo, os modelos de prioridades descritos no Real-time CORBA são bastante úteis nas estratégias de conversão do sistema legado para o sistema tempo real. O modelo de

prioridades SERVER_DECLARED permite ao servidor ditar a prioridade na qual uma invocação feita a um determinado objeto deve ser executada. Neste modelo o servidor designa a prioridade à priori quando o objeto é ativado. A prioridade é codificada dentro da referência ao objeto, que é publicada para o cliente na referência do mesmo. Podem ser utilizados alguns mapeamentos de prioridades dependendo da implementação de ORB de tempo real usado. Neste caso específico é utilizado o mapeamento linear. Neste mapeamento, cada prioridade nativa individualmente é convertida para um intervalo contínuo de prioridades CORBA. Desta forma todo intervalo de prioridades CORBA é utilizado no mapeamento, sendo a prioridade máxima configurada igual a 32.767, ou ainda, podendo ser representada por *RTCORBA::maxPriority.* Neste caso, através do mapeamento de prioridades, esta última é convertida para a máxima prioridade do sistema operacional.

2.1.3 Timeouts

Diversas operações devem limitar o intervalo de tempo gasto esperando que o ORB e o servidor processem as solicitações. Esta característica é importante para que seja possível identificar servidores que estejam bloqueados e tomar ações apropriadas, como, por exemplo, migrar as ações para servidores alternativos ou responder a uma possível falha de sistema que requer reparo automático ou manual. Em todos os casos, é desejável finalizar as invocações de operações após ter decorrido o tempo máximo para a execução.

O suporte ao uso de *timeouts* nas operações é realizado pelo CORBA Messaging. Tal especificação inclui políticas que podem ser configuradas para automaticamente encerrar as invocações de operações cujo tempo foi excedido. No total são dez políticas, conforme podem ser observadas em [5]. Para as estratégias definidas neste artigo, utiliza-se a política *RelativeRoundtripTimeout* que afeta apenas o comportamento dos clientes, ou seja, nenhuma informação de tempo é passada para o servidor.

No início de cada invocação de operação, o ORB cliente envia um *RelativeRoundtripTimeoutPolicy* definido na interface *Messaging*. Caso esta política seja configurada na aplicação, o *timeout* associado será usado para limitar o tempo gasto para estabelecer a conexão ao servidor remoto, enviar solicitação ao servidor remoto e esperar pela resposta. Antes do ORB esperar por uma operação de bloqueio, ele computa o tempo gasto desde o início da solicitação e inicia o valor de *timeout* para o tempo restante especificado pela política. Por exemplo, se o valor total do *timeout* é 15 msecs e o ORB leva 5 msecs para estabelecer a conexão, apenas 10 msecs estarão disponíveis para enviar as solicitações e receber a reposta.

Caso o *timeout* expire durante qualquer ponto na seqüência de processamento, o ORB cliente liberará os recursos usados para a solicitação e emitirá uma exceção do tipo CORBA::TIMEOUT. Devido aos atrasos de rede ou porque o servidor levou mais tempo que o esperado para processar a solicitação, é possível que a reposta chegue após o ORB cliente ter gerado a exceção do tipo CORBA::TIMEOUT, cabendo a aplicação tomar providências adequadas diante de tal situação.

Para exemplificar como a definição dos *timeouts* acontece na prática, considera-se uma operação de navegação, onde se deseja obter o próximo elemento de um vetor de elementos armazenado no objeto B. Para realizar uma navegação para o próximo elemento do vetor, o objeto A requisita ao objeto B, através do método **Proximo(),** a realização de tal ação. O servidor do objeto B, por sua vez, deverá atender a esta solicitação em um intervalo de 2 (dois) segundos. Caso isto não ocorra será lançada uma exceção do tipo TIMEOUT pelo cliente e o servidor deverá ignorar tal ação, devendo o cliente refazer a sua solicitação, após receber um alerta através da sua interface gráfica.

A política *RelativeRoundtripTimeout* pode ser usada no nível do ORB, no nível de *thread* ou no nível de objeto. Definir as políticas de qualidade de serviços no nível do ORB significa que

o cliente pode sobrescrever as características originais do nível de sistema, de forma a controlar a qualidade de serviços para todas as solicitações realizadas através daquele ORB [5]. Também podem ser utilizadas políticas de qualidade de serviços no nível de *thread*. Estas podem sobrescrever aquelas do nível de sistema e do ORB. Ela oferece uma granularidade de controle mais fina, permitindo que as solicitações feitas em uma dada *thread* tenham diferentes características de qualidade de serviços das solicitações feitas por outras *threads* no mesmo processo de aplicação. Ainda assim, ao se utilizar estas políticas no nível de ORB, pode haver efeitos indesejáveis, e para reduzi-los, tal política pode ser sobrescrita no nível do objeto, que oferece o controle de granularidade mais fino suportado pelo OMG CORBA Messaging.

Após a inicialização ORB, pode ser realizada a inicialização específica da aplicação. Em princípio deve ser obtida a referência para o objeto desejado através do serviço de nomes. Para efeitos explicativos, o objeto B obtido a partir do serviço de nomes é denominado `objetob`. Todas as alterações realizadas em relação aos *timeouts* são efetuadas no código do cliente ao acessar os objetos críticos no sistema. Desta forma, os clientes devem passar toda a configuração dos *timeouts* durante a solicitação da operação no objeto servidor, já que todas as características são apenas salientadas no lado do cliente.

Sendo assim, estas alterações só podem acontecer nas chamadas executadas a partir de um cliente com suporte a tempo real. Caso esta chamada seja executada a partir de um objeto cliente do sistema legado, ou seja, implementado sobre um ORB sem suporte a tempo real, será necessária a definição de um objeto adaptador, sobre o ORB de tempo real, que deve direcionar as chamadas dos objetos não-tempo real aos objetos de tempo real, configurando todas as requisições de tempo e prioridades com que aquele método deve ser executado no ORB de tempo real.

Na prática, o objeto não tempo real chama o objeto adaptador, que por sua vez ,configura os parâmetros necessários para a configuração de tempo real e só então é que realiza a chamada ao método no objeto B de tempo real, conforme observado na Figura 3.

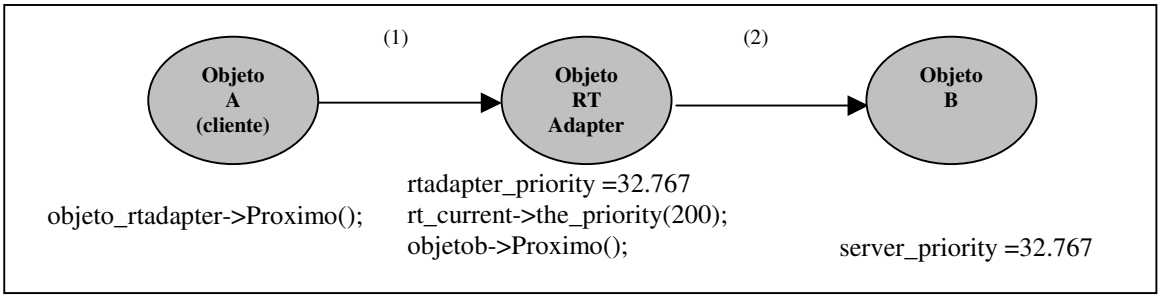

Figura 3 Modelo de Prioridades SERVER_DECLARED

Na Figura 3, em (1) o objeto A realiza a chamada do método através do objeto RT-Adapter a uma prioridade qualquer. Como o objeto RT-Adapter foi configurado durante a inicialização no modelo declarado no servidor, utilizando a prioridade 32.767, esta será, então, a prioridade de execução deste método. Em seguida, este último realiza a chamada ao objeto B a uma prioridade qualquer (2), que não é utilizada, já que o servidor de slides está configurado como o modelo de prioridades declarado no servidor executando aquele método na sua prioridade originalmente configurada, que no caso acima corresponde à máxima prioridade CORBA, 32.767.

Através da realização de todas estas etapas, pode ser obtida a estratégia 1, definida neste trabalho. A seqüência de procedimentos que devem ser realizados para obtê-la, é descrita na Figura 4.

Figura 4 Seqüência de Passos para Implantação da Estratégia 1

2.2 Estratégia 2

Esta segunda estratégia utiliza o modelo de prioridades CLIENT_PROPAGATED em conjunto com a política *RelativeRoundtripTimeout* definida no CORBA Messaging. Nesta estratégia não são utilizados os *threadpools*.

2.2.1 Modelo de Prioridades

No modelo de prioridades declarado no cliente, a prioridade CORBA do cliente é transmitida ao longo da solicitação. Cada servidor mapeia a prioridade CORBA do cliente para a prioridade nativa apropriada à plataforma do *host*.

Neste modelo, cada invocação transporta a prioridade CORBA da operação na lista de contexto de serviços que está incluída como parte da solicitação GIOP. Cada ORB do cliente e do servidor, respectivamente, mapeia esta prioridade CORBA para a prioridade nativa do sistema operacional e processa as solicitações com esta prioridade.

Para um sistema legado que precisa ter requisições que atendam a *deadlines* estritos, podem-se identificar as operações críticas nos seus objetos a partir da identificação das suas prioridades. Para os objetos que possuem operações que devem ser executadas nas maiores prioridades do sistema, a política CLIENT_PROPAGATED é configurada no servidor e exportada para o cliente através da referência do objeto. Este, por sua vez, irá realizar a requisição ao método crítico e antes disto, dita a prioridade que tal chamada deve possuir, devendo esta ser respeitada pelo servidor.

Inicialmente são efetuadas as alterações no código do servidor para que o mesmo seja inicializado utilizando o modelo de prioridades desejado, conforme definido na especificação do Real-time CORBA.

A prioridade associada ao objeto crítico neste modelo, no momento da inicialização, é uma prioridade intermediária entre a maior e a menor do sistema. Isto ocorre porque os métodos associados àquele objeto, que devem atender a restrições de tempo e prioridades, passam na sua requisição a prioridade máxima para execução, *RTCORBA::maxPriority*, enquanto os demais métodos devem ser executados a uma prioridade superior às demais do sistema, sendo configurada na inicialização do objeto, que como no caso acima, é a prioridade CORBA de 20.000, que é mapeada para a prioridade do sistema operacional correspondente. Aqui novamente é utilizado o mapeamento de prioridades linear. Embora os servidores determinem quando um objeto é criado com o modelo de prioridades CLIENT_PROPAGATED, os clientes em última instância é que determinam a prioridade CORBA na qual as operações a um objeto são invocadas.

Todas as operações realizadas até o momento referem-se a inicialização do servidor. O código do cliente deve conter as alterações realizadas nas prioridades das operações para o modelo de prioridades utilizado, bem como toda a configuração dos *timeouts*.

É importante salientar que estas alterações só podem acontecer nas chamadas executadas a partir de um cliente com suporte a tempo real. Caso esta chamada seja executada a partir de um objeto cliente do sistema legado, ou seja, implementado sobre um ORB sem suporte a tempo real, será necessária a definição de um objeto adaptador, sobre o ORB de tempo real que direciona as chamadas dos objetos não-tempo real aos objetos de tempo real configurando todas as requisições de tempo e prioridades com que aquele método deve ser executado no ORB de tempo real.

Para selecionar a prioridade CORBA na qual uma operação é invocada, um cliente utiliza um objeto *RTCORBA::Current.* Uma situação real pode ser observada na Figura 5.

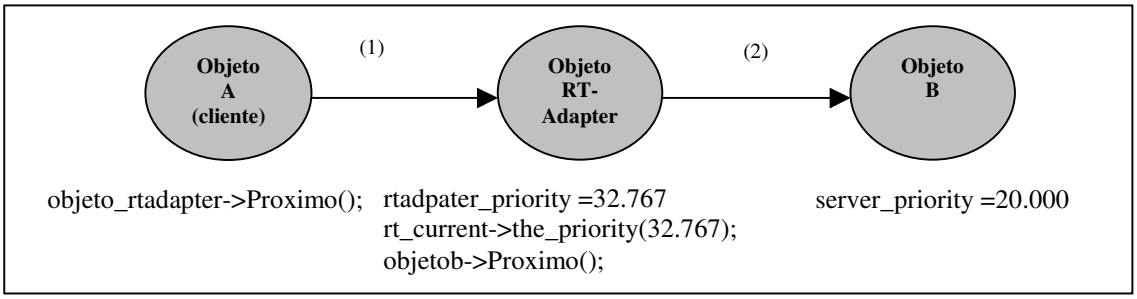

Figura 5 Modelo de Prioridades CLIENT_PROPAGATED

Em (1) ocorre a chamada ao método **Proximo()** do RT-Adapter que é executado na prioridade de 32.767, que corresponde à máxima prioridade CORBA. Em seguida, o método **Proximo()** do objeto B é chamado na prioridade propagada durante a sua requisição, representada em (2), no valor 32.767 que é a prioridade de execução do método no objeto B. Esta prioridade sobrescreve a prioridade 20.000 definida na inicialização de B.

De acordo com os desafios de projeto, é necessário incluir características temporais na utilização de determinados objetos. O suporte para a utilização destes *timeouts* é dado pela especificação do CORBA Messaging, conforme citado anteriormente

2.2.2 Timeouts

As alterações a serem realizadas em nível de código com relação aos *timeouts* devem ser feitas no código do cliente, pois estas políticas são associadas aos clientes dos objetos. As mesmas alterações observadas na estratégia 1, com relação a configuração dos *timeouts* das operações, são efetuadas também aqui nesta estratégia. No entanto, o resultado esperado é um pouco diferente dado que os modelos de prioridades utilizados para os dois casos são diferentes e apenas um deles utiliza a idéia de *threadpools.* No que diz respeito às políticas de *timeouts,* elas são as mesmas para ambos os casos, pois limitam o tempo de requisição aos métodos considerados críticos.

Assim, tomando a operação de navegar ao próximo elemento, é definido um *timeout* de 2 segundos para que o método seja executado pelo servidor, porém da mesma forma que para a estratégia 1, este *timeout* não pode ser iniciado no cliente final, pois o mesmo está sobre um ORB sem suporte a tempo real, que não apresenta tal recurso, assim deve ser iniciado a partir do RT-Adapter. Estas mesmas alterações são replicadas para todas as ações dos objetos que possuem ações prioritárias e com controles temporais.

A seqüência de procedimentos resumindo a estratégia 2 é apresentada na Figura 6.

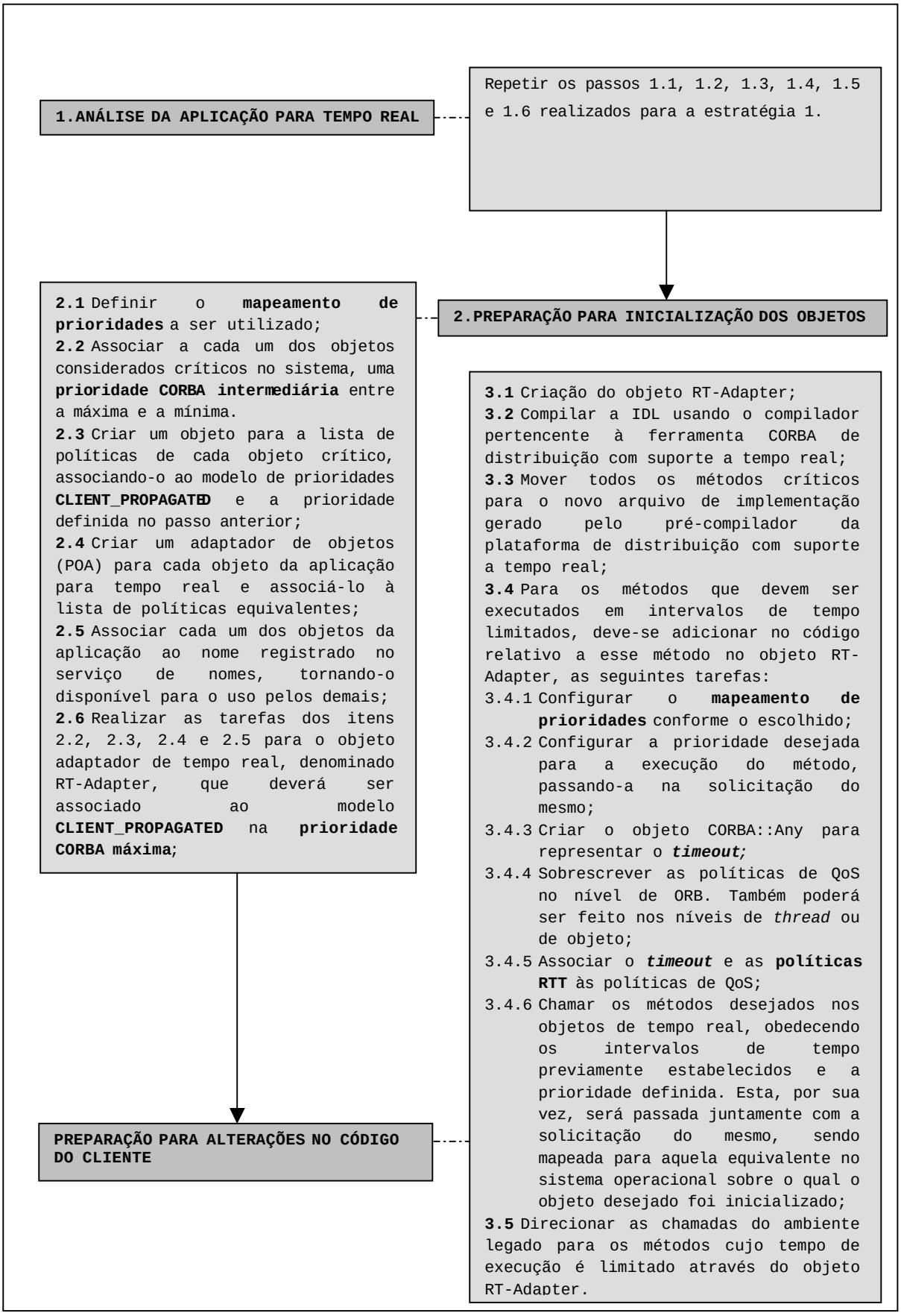

Figura 6 Seqüência de Passos para Implantação da Estratégia 2

3 Estudo de Casos de uma Aplicação de Tempo Real

Esta seção tem a finalidade de validar as estratégias de integração definidas anteriormente e para isto é utilizado o sistema legado de co-autoria de aulas. Este sistema é acrescido de alguns aspectos não funcionais que beneficiarão as questões temporais, transformando-o em um sistema de tempo real *soft*. Conforme citado, a opção por um sistema legado tem o objetivo de mostrar a capacidade de migração de um ambiente para outro, onde o último leva em consideração novos aspectos de qualidade de serviços, previsibilidade, melhoria de desempenho, dentre outras características de sistemas de tempo real.

Provavelmente todos os sistemas possuem suas entidades críticas que devem ser atendidas com maior prioridade, tempo de resposta mais preciso e maior qualidade de serviços. Por outro lado, grande parte dos sistemas possuem módulos que não são de tempo real onde não é necessária esta tentativa de atender a todas as solicitações em tempo e prioridade estritos.

3.1 A Aplicação Original

A ferramenta de co-autoria foi desenvolvida utilizando a tecnologia CORBA. Na sua versão original [8], o sistema de co-autoria possui todos os seus objetos implementados sobre um ORB sem suporte a tempo real.

O cenário da aplicação possui autores que trabalham na construção de uma aula formada por um conjunto de slides. Através deste ambiente, tais autores têm uma visão única do material criado. Os slides são criados e editados pelos autores que participam das sessões, porém com a restrição de que apenas um autor pode criar, editar ou excluir slides cada vez. Este é considerado editor e tem toda ação de navegação entre os slides replicada para os demais autores presentes na seção, e toda edição sobre os slides deve ser ratificada pelos mesmos autores, através de uma votação majoritária. Todo autor deve ter direito a tornar-se o editor da aula. Para garantir isto, quando um autor que não possuir a permissão de edição da aula quiser realizar alguma alteração em um determinado slide, o mesmo deverá requisitar esta permissão para si. Ao realizar tal requisição, este autor é inserido em uma fila de espera, que determinará a ordem de acesso à edição da aula.

No sistema de co-autoria existem três objetos principais que são implementados e distribuídos sobre o Orbacus: a) o Autor, que instanciado representa cada usuário do ambiente. Este componente tem, basicamente, como principais funções enviar e receber o slide que está sendo trabalhado pelo editor, emitir opiniões sobre edições e também sobre entrada de novos usuários no sistema; b) o Coordenador, que é responsável pelo controle da consistência da aula guardando informações como o nome de todos os autores presentes no ambiente, o editor da aula e a fila dos autores que estejam requerendo o acesso à edição da aula em um dado instante. O coordenador gerencia também o processo de votação sobre a edição dos slides, emitindo telas para voto dos demais autores, contabilizando o resultado; c) o Servidor de Slides, que representa o conjunto de slides que compõem a aula. Suas funções basicamente refletem a navegação e edição sobre este conjunto, como ir para o próximo slide, ir para o último slide, inserir novo slide, etc.

O sistema em questão utiliza o serviço de eventos CORBA, para permitir a comunicação entre os elementos. Através deste serviço, são utilizados canais de eventos para a transmissão dos pedidos para votação, notificações sobre saídas ou desconexões de autores no ambiente, atualização da lista de usuários presentes e que estão na lista de acesso à permissão de edição da aula e transmissão do slide a ser mostrado para todos os participantes da aula.

3.2 Aspectos de Tempo Real

Na aplicação de co-autoria em tempo real, alguns desafios de projeto foram lançados, tendo como objetivo atender algumas operações previamente observadas como críticas através do uso intenso do sistema em um ambiente de rede de alta velocidade.

Considerando estes fatores, verifica-se que apenas alguns módulos do sistema de co-autoria necessitam de uma migração para tempo real. Sendo assim, este sistema deve ter parte dos seus objetos convertidos para sem utilizados sobre o TAO, ORB com suporte ao Real-time CORBA e outra parte permanece implementada sobre o ORB Orbacus [12], conforme o sistema original.

Para se decidir quais dos objetos devem ser convertidos observa-se as operações do sistema:

- O objeto Servidor de Slides tem a maioria das suas operações abrangendo a navegação e a edição de slides. Estas operações são identificadas como críticas e assim, é necessário um tempo de resposta mais preciso para elas, ao mesmo tempo em que devem ser executadas com uma prioridade maior dentro do sistema.
- O objeto Coordenador fica responsável pelos processos de votação, controle de entrada e saída de usuários no sistema e a passagem da permissão da edição da aula entre os participantes. Observa-se que o processo de votação deve ser um processo prioritário. O controle da permissão é prioritário em parte, porque só o será no momento exato onde a passagem da permissão está sendo feita de um autor para outro, pois o sistema deve permanecer o menor tempo possível sem editor. O controle de entrada e saída não é crítico, pois o ideal é que o sistema processe as informações de edição da aula entre os seus participantes com maior prioridade que manipule entrada e saída de usuários no sistema.
- O objeto Autor, possui a maioria das suas ações associadas à interface gráfica. Esta apresenta janelas onde o autor digita as informações que deseja incluir nos slides e faz a sua navegação. Os votos solicitados pelo objeto Coordenador são inseridos a partir destas janelas. As janelas de votação enviadas para o autor possuem um controle temporal controlado pela própria interface gráfica, onde o autor tem disponível 15 segundos para inserir o seu voto. Desta forma, não há necessidade de seus métodos utilizarem um ORB com suporte a tempo real para controlar o processo de votação.

Baseado na descrição das operações verifica-se que os objetos Servidor de Slides e Coordenador estão associados às várias operações que devem envolver *timeouts* e alta prioridade, precisando então, de maior qualidade de serviços. Estes objetos devem ser convertidos para um ORB com suporte a tempo real, o TAO, enquanto o objeto Autor permanece sobre o Orbacus.

A operação para cadastro de slides é ilustrada na Figura 7 para o caso de ser utilizada a estratégia 1. Para esta estratégia se utilizam dois *threadpools*, onde um deles está associado ao objeto Servidor de Slides e o outro ao Coordenador. Ambos os objetos são considerados críticos e utilizam o modelo de prioridades declarado no servidor.

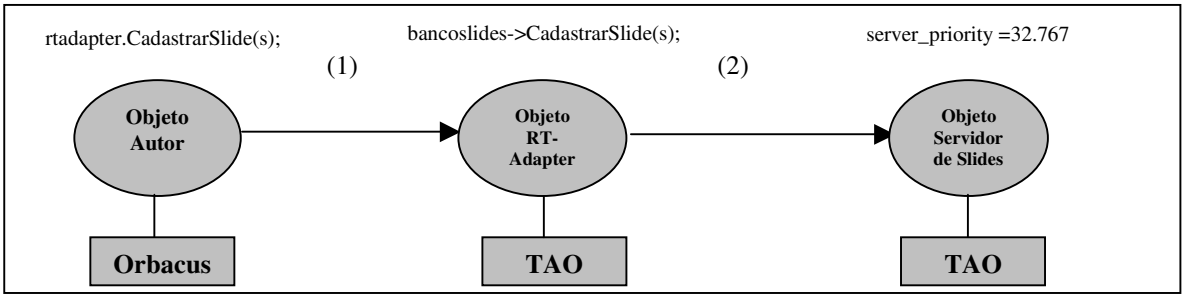

Figura 7 Cadastro de Slides usando Estratégia 1

Na Figura 7, em (1) acontece a chamada do cliente, através da interface gráfica do Autor, ao método *CadastrarSlide* após o editor digitar as informações que deseja inserir no slide. Esta chamada é feita ao método *CadastrarSlide* do objeto RT-Adapter. Este, por sua vez, insere restrições de tempo e chama o método *CadastrarSlide* do Servidor de Slides numa prioridade qualquer. Este valor de prioridade não importa, pois o método será executado na prioridade 32.767 definido previamente na criação do objeto Servidor de Slides, já que é utilizado o modelo SERVER_DECLARED.

Considerando-se a mesma operação de cadastro de slides, usando a estratégia 2, o processo inicialmente ocorre de forma semelhante, porém no momento em que o objeto RT-Adapter repassa a solicitação do método *CadastrarSlide* do objeto Servidor de Slides, além de configurar as restrições de tempo, configura também a prioridade CORBA na qual o método solicitado deve ser executado. Isto acontece, pois tal objeto, durante a sua inicialização foi configurado com o modelo de prioridades propagado pelo cliente. Sendo assim, a prioridade propagada será mapeada para a prioridade equivalente no sistema operacional sobre o qual o objeto está sendo executado e sobrescreverá a prioridade configurada na inicialização do mesmo.

4 Testes e Resultados Obtidos

Na realização dos teste convencionou-se a distribuição de objetos utilizando três máquinas numa rede de alta velocidade. Nestas máquinas estão distribuidos os objetos Servidor de Slides, Coordenador, RT-Adapter, os canais de eventos e o serviço de nomes, além do objeto Autor representando cada usuário no ambiente de co-autoria de aulas.

Basicamente foi utilizado o objeto Servidor de Slides para a obtenção destes resultados com suas funções de navegação. Esta escolha é realizada, pois as funções de votação, dentre outras operações do objeto Coordenador, exigem uma interação com o usuário, o que pode mascarar os testes, pois insere mais uma variável ao sistema, que é o tempo do Autor/Usuário responder a uma votação, por exemplo. Desta forma, são escolhidas as operações de cadastro de slides e navegação ao próximo slide para a obtenção dos resultados, sendo que a primeira delas é realizada quando existe apenas o editor no sistema, pois se houvessem mais usuários seria necessário realizar uma votação e dependeria do tempo no qual os demais autores votassem contra ou a favor da inserção do slide. A operação de navegação ao próximo é realizada considerando três usuários.

Foram consideradas três aplicações:

- A ferramenta de co-autoria original, sem a integração de aspectos de tempo real;
- A ferramenta de co-autoria integrada a aspectos de tempo real utilizando a estratégia 1;
- A ferramenta de co-autoria integrada a aspectos de tempo real utilizando a estratégia 2.

Todas as três aplicações foram submetidas a dois cenários de tráfego. O **primeiro cenário** considera o ambiente de rede sujeito apenas ao tráfego dos dados trocados entre os objetos do sistema de co-autoria. No **segundo cenário** foi considerado um tráfego com relação ao ambiente de videoconferência associado às ferramentas do Mbone [13] para troca de opiniões sobre as edições e votações realizadas sobre a aula, ou seja, além do tráfego referente aos objetos do sistema de co-autoria, existe aquele imposto pelo ambiente de videoconferência.

A análise é feita baseada no tempo de resposta de algumas solicitações aos objetos. Os valores de intervalos de tempo de reposta às operações para as duas situações de carga na rede são medidos em segundos. Estes intervalos de tempo são obtidos a partir do RTT (Round Trip

Timeout) relativo, ou seja, o tempo gasto para uma solicitação a um método de um objeto ir até o seu servidor e retornar com a resposta para o cliente que a solicitou.

Os valores relativos ao tempo médio gasto para o cadastramento de um slide podem ser observados na Tabela 2, para os dois cenários propostos. Os valores obtidos com a navegação ao próximo slide são mostrados na Tabela 3.

Estes valores são obtidos pela média entre os dados observados com uma massa de dados de 200 cadastramentos e 200 navegações. Para a obtenção destes valores médios é utilizado o software Statistcs, que calcula, além da média, o desvio padrão, a variância, os valores mínimo e máximo obtidos. As colunas Tempo Real 1 e Tempo Real 2 referem-se, respectivamente aos resultados obtidos se aplicando as estratégias 1 e 2.

	Não Tempo Real	Tempo Real 1	Tempo Real 2
Cenário 1	1.00362 s	0.02589 s	0.02378 s
Cenário 2	6.04025 s	0.07101 s	0.07481 s

Tabela 2 - Tempo Médio de Resposta para Cadastramento

	Não Tempo Real	Tempo Real 1	Tempo Real 2
Cenário 1	2.99053 s	0.06845 s	0.06861 s
Cenário 2	6.11601 s	0.11241 s	0.11837 s

Tabela 3 - Tempo Médio de Resposta para Acesso ao Proximo

Observando as tabelas com os intervalos de tempo em segundos, nota-se que para a aplicação utilizada neste estudo de caso, ambas as estratégias foram eficientes, pois conseguiram reduzir o tempo de resposta aos objetos, obedecendo aos *deadlines* impostos pela aplicação quando a mesma foi integrada a aspectos de tempo real, tornando-a um sistema de tempo real *soft.*

5 Conclusões

Este artigo abordou as características de um novo sistema integrado com aspectos de tempo real, através da aplicação de duas estratégias aqui definidas para atender aos desafios de um projeto de uma aplicação legada com requisitos de tempo real.

Em seguida foi descrita uma aplicação exemplo de um sistema legado de co-autoria de aulas, no qual foi observada a necessidade de conversão de alguns de seus objetos e operações para um ambiente de tempo real, sendo então projetada uma nova aplicação integrada a aspectos de tempo real, com o objetivo de atender aos requisitos estabelecidos. Desta forma, a aplicação de co-autoria sofreu o processo de integração de características de tempo real através do uso das estratégias 1 e 2 para tornar-se o sistema desejado.

A escolha por duas estratégias se deu, principalmente, devido aos modelos de prioridades definidos na especificação do Real-time CORBA. Como existem dois modelos e cada um deles possui as suas peculiaridades, optou-se por utilizar uma estratégia para cada um, observando que ambas possuem desempenho satisfatório.

A principal contribuição deste artigo foi a definição das estratégias que utilizam recursos das especificações do Real-time CORBA e CORBA Messaging. Através destas estratégias é possível a obtenção de um sistema híbrido, onde alguns módulos apresentam suporte a tempo real, enquanto outros operam sem este suporte, mostrando-se como o sistema original. Para que

cada uma delas seja empregada, foi necessário o desenvolvimento de um protótipo com a finalidade de mostrar como os desafios de projeto foram atingidos.

Desta forma conclui-se que é possível transformar um sistema distribuído legado em um sistema de tempo real, utilizando os recursos definidos nas especificações do Real-time CORBA e CORBA Messaging, de forma gradual e harmônica, sem causar tanto impacto durante a transição.

Como trabalho futuro, as estratégias aqui definidas podem ser aplicadas a um sistema distribuído legado que utilize objetos que trocam dados na forma de mídias contínuas. Poderá ser obtido melhor desempenho durante a utilização de uma das duas estratégias, já que trafegam dados de áudio e vídeo entre os objetos distribuídos. Também pode ser sugerido como trabalho futuro o desenvolvimento de uma ferramenta de engenharia de software para automatizar o processo de integração do ambiente distribuído legado com aspectos de tempo real.

6 Referências Bibliográficas

[1] BENNETT, K. Legacy Systems: Coping With Success. **IEEE Software**, vol.12, n.1, jan. 1995, pp 19-23.

[2]OBJECT MANAGEMENT GROUP. **The Common Object Request Broker: Architecture and Specification Revision 2.6**, OMG Technical Document formal/01-02-33, dez. 2001. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/01-12-35>. Acesso em 30 mar. 2002.

[3] OBJECT MANAGEMENT GROUP. **CORBA Success Stories.** Disponível em: <http://www.corba.org/success.htm >Acesso em 28 mar. 2002.

[4] OBJECT MANAGEMENT GROUP. **Real-Time CORBA**, OMG TC Document formal/01-12-28, dez. 2001. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/01-12-28>. Acesso em 30 mar. 2002.

[5] OBJECT MANAGEMENT GROUP. **CORBA Messaging Specification,** OMG TC Document orbos/98-05-05, maio. 1998. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/01-12-26>. Acesso em 30 mar. 2002.

[6] CURREY, Jon. **Real-Time CORBA Theory and Practice : A Standards-Based Approach to Development of Distributed Real-Time Systems**. In: EMBEDDED SYSTEMS CONFERENCE, San Jose 2000. Disponível em: <http://www.highlander.com/white_papers/abstracts.html#RealTimeCORBATandP>. Acesso em 30 mar. 2002.

[7] SCHIMIDT, Douglas; VINOSKI, Steve. Object Interconnections: Real-time CORBA, Part 2: Applications and Priorities. **C/C++ Users Journal**, jan. 2002. Disponível em: <http://www.cuj.com/experts/2001/vinoski/vinoski.htm?topic=articles>. Acesso em 30 mar. 2002.

[8] TRINTA, Fernando. **Arquiteturas Distribuídas para Co-autoria Cooperativa de Aulas na Internet**. Recife, 2000. 131f. Dissertação (Mestrado em Informática) - Centro de Informática, Universidade Federal de Pernambuco.

[9] BARROS, Juliana Diniz; FERRAZ, Carlos André. Plataformas Distribuídas para Aplicações de Tempo Real. **Anais** do WORKSHOP DE TEMPO REAL, 3, 2001, Florianópolis. Universidade Federal de Santa Catarina, 2001. 1 CD-ROM.

[10] OBJECT COMPUTING INC. **TAO Developers Guide.** 1 ed. St Louis:[s.n], 2000.730p.

[11] SCHIMIDT, Douglas; VINOSKI, Steve. Object Interconnections: Real-time CORBA, Part 3: Thread Pools and Synchronizers. **C/C++ Users Journal**, mar. 2002. Disponível em: <http://www.cuj.com/experts/2003/vinoski.htm?topic=articles>. Acesso em 30 mar. 2002.

[12] IONA. Orbacus ORB. Disponível em: <http://www.orbacus.com/>. Acesso em 28 mar. 2002.

[13] UCL Network and Multimedia Research Group. **Mbone Conference Applications**. Disponível em: <http://www-mice.cs.ucl.ac.uk/multimedia/software>. Acesso em 28 fev. 2002.