UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma abordagem de escalonamento adaptativo no ambiente Real-Time CORBA

por

ALEXANDRE CERVIERI

Dissertação submetida à avaliação, como requisito parcial
para obtenção do grau de
Mestre em Ciência da Computação.

Prof. Dr. Rômulo Silva de Oliveira
Orientador

Prof. Dr. Cláudio Fernando Resin Geyer
Co-orientador

Porto Alegre, abril de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Cervieri, Alexandre

Uma abordagem de escalonamento adaptativo no ambiente Real-Time CORBA / por Alexandre Cervieri. – Porto Alegre: PPGC da UFRGS, 2002.

113 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientador: Oliveira, Rômulo Silva de; Co-orientador: Geyer, Cláudio Fernando Resin.

1. Sistemas distribuídos de tempo real. 2. Escalonamento adaptativo. 3. RT-CORBA. 4. TAO. I. Oliveira, Rômulo Silva de. II. Geyer, Cláudio Fernando Resin. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof ª Wrana Maria Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Prof. Jaime Evaldo Fensterseifer
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Haro

Agradecimentos

Antes de adaptar o pouco do texto desta dissertação que eu já tinha escrito ao formato que realmente ele deveria estar vi quais eram as principais partes que deveriam constar no trabalho. Além de todos os itens obrigatórios e amplamente conhecidos por todos, como introdução e conclusões, havia um item, opcional, intitulado "Agradecimentos".

O item é opcional, porém, vejo-o na verdade como um item mais do que obrigatório, essencial. Apesar dos defeitos que ainda podem acompanhar esta dissertação é importante sentir um pouco de orgulho do resultado final de um tempo de muito estudo e trabalho. Entretanto, mais importante do que isso é saber valorizar e agradecer todas as pessoas que contribuíram de alguma forma para a conclusão desta dissertação.

Em primeiro lugar, quero não só agradecer, mas dedicar esta dissertação, assim como fiz com meu trabalho de conclusão de graduação, a meus pais – que sempre me ensinaram a lutar pelos meus sonhos e eis mais um deles conquistado.

Devo muito também ao professor Rômulo, meu orientador, sem seu auxílio e orientações nos momentos de dúvida seria impossível ter concluído esta dissertação. Todos que me conhecem sabem que não sou muito bom com o português e escrever em inglês também não é meu forte, por isso, também tenho que agradecer muito a paciência que minha namorada, Paula Berger, teve ao fazer importantes correções de português e de inglês (se por acaso existirem muitos erros de português nestes agradecimentos, a Paula não tem culpa, não passei essa parte para ela corrigir).

Finalmente, queria agradecer ao meu co-orientador prof. Cláudio Geyer. E também a lista tao-users pelas respostas a alguns dos problemas que tive com o TAO.

Sumário

Lista de Figuras **6**

Lista de Tabelas **7**

Resumo **8**

Abstract **9**

1 Introdução **10**

2 Mecanismos de Adaptação em Aplicações de Tempo Real **12**

2.1 Caracterização de sistemas de tempo real **12**

2.2 Classificação das abordagens de escalonamento de tempo real **13**

2.3 Cenários para adaptação de aplicações de tempo real **15**

2.4 Mecanismos de adaptação em aplicações de tempo real **16**

2.5 Propostas de Adaptação na Literatura **20**

2.6 Considerações Finais **22**

3 CORBA **23**

3.1 OMG – Object Management Group **23**

3.2 OMA – Object Management Architecture **25**

3.3 ORB – Object Request Broker **27**

3.3.1 IDL – Interface Definition Language 28

3.3.2 Invocação estática de métodos 30

3.3.3 Invocação dinâmica de métodos 31

3.3.4 Object Adapter 32

3.4 CORBA *Services* **34**

3.4.1 Naming Service 36

3.4.2 Event Service 37

3.5 CORBA *Facilities* e CORBA *Domains* **38**

3.6 Considerações Finais **39**

4 RT-CORBA e o ORB TAO **41**

4.1 Visão Geral do padrão RT – CORBA **42**

4.2 Controle de Recursos **43**

4.2.1 Threads 43

4.2.2 Estoque de Threads (Thread Pools) 44

4.2.3 Prioridades 45

4.2.4 Fila de requisições e controle de fluxo 47

4.2.5 Transporte 47

4.3 Interceptors **50**

4.3.1 Request Interceptors 52

4.4 Mecanismos de Sincronização 54
4.5 Serviço de Escalonamento do RT-CORBA 56
4.6 Estudo de caso: TAO 57
4.6.1 Visão Geral 58
4.6.2 Subsistema de E/S 60
4.6.3 Módulo principal do ORB 61
4.6.4 Protocolo GIOP de tempo real 62
4.6.5 Serviço de Escalonamento de Tempo Real 63
4.6.6 Serviço de Eventos de Tempo Real 67
4.6.7 Object Adapter de Tempo Real 70
4.6.8 Compilador IDL e camada de apresentação 72
4.6.9 Otimizações no gerenciamento de memória 73
4.6.10 Especificação de QoS 74
4.7 Considerações Finais 77
5 Mecanismo de Adaptação Proposto 78
5.1 Descrição 78
5.2 Implementação do mecanismo de adaptação no contexto do ORB TAO 81
5.2.1 Task 82
5.2.2 PeriodicTask 82
5.2.3 AdaptiveService 84
5.3 Considerações Finais 86
6 Experiências 87
6.1 Estrutura da aplicação experimental 87
6.2 Cenário das experiências 88
6.3 Resultados obtidos 90
6.3.1 Experiências preliminares com o mecanismo de adaptação proposto 94
6.3.2 Resultados com o mecanismo de adaptação proposto 96
6.4 Considerações Finais 98
7 Conclusões e Trabalhos Futuros 99
Anexo 1 Instalação do ACE/TAO 102
Anexo 2 Instalação da Aplicação Experimental 104
Bibliografia 109

Lista de Figuras

FIGURA 2.1 - Classificação das Abordagens para Escalonamento de Tempo Real. 14
FIGURA 2.2 - Efeito da adaptação na área de aceitação no espaço de recursos. 16
FIGURA 2.3 - Mecanismo básico de adaptação. 17
FIGURA 3.1 - Modelo estendido da OMA. 27
FIGURA 3.2 - Estrutura de um ORB. 28
FIGURA 3.3 - Estruturas comumente encontradas em IDL. 29
FIGURA 3.4 - Diagrama de interação típico. 30
FIGURA 3.5 - Ciclos de vida de objetos CORBA e servants. 34
FIGURA 3.6 - Alternativas de interação para troca de eventos. 38
FIGURA 3.7 - Relação das especificações da OMG. 39
FIGURA 4.1 - Principais componentes do RT-CORBA. 42
FIGURA 4.2 - Mapeamento de prioridades CORBA para prioridade nativas. 45
FIGURA 4.3 - Modelo de prioridade definida pelo servidor. 46
FIGURA 4.4 - Modelo de prioridade definida pelo cliente. 46
FIGURA 4.5 - Configuração e seleção de protocolos. 48
FIGURA 4.6 - Priority Banded Connections e Private Connections. 49
FIGURA 4.7 - Elementos da arquitetura do ORB TAO. 59
FIGURA 4.8 - Componentes do módulo principal do ORB TAO. 61
FIGURA 4.9 - Componentes do Serviço de Escalonamento do TAO. 64
FIGURA 4.10 - Passos executados durante uma execução de configuração. 66
FIGURA 4.11 - Arquitetura do Serviço de Eventos de Tempo Real. 68
FIGURA 4.12 - Mecanismos de despacho do Object Adapter do TAO. 70
FIGURA 4.13 - Estratégias de demultiplexação otimizadas. 72
FIGURA 5.1 - Mecanismo básico de realimentação. 78
FIGURA 5.2 - Mecanismo de adaptação proposto. 80
FIGURA 5.3 - Diagrama da implementação do modelo de adaptação. 81
FIGURA 5.4 - Coleta de informações e atuação do Serviço de Adaptação. 85
FIGURA 6.1 - Estrutura possível de um sistema de automação industrial. 88
FIGURA 6.2 - Evolução do DY(t) ao longo do tempo para o sistema sem adaptação. ..91
FIGURA 6.3 - Evolução do DY(t) ao longo do tempo com a primeira forma de adaptação (i). 93
FIGURA 6.4 - Evolução do DY(t) ao longo do tempo com a primeira forma de adaptação (ii). 93
FIGURA 6.5 - Evolução DY(t) para adaptação com SDY=-50000us e K_P=1/250000. ..94
FIGURA 6.6 - Evoluação de DY(t) com SDY=-60000us e K_P=1/420000. 97

Lista de Tabelas

TABELA 2.1 - Análise qualitativa de algumas propostas de técnicas adaptativas da literatura. 21
TABELA 3.1 - Grupos de membros da OMG. 24
TABELA 3.2 - Processo para invocação dinâmica de métodos. 31
TABELA 3.3 - CORBA *Services* organizados segundo a ORBOS. 35
TABELA 4.1 - Definição da interface do Serviço de Escalonamento em IDL. 57
TABELA 4.2 - Interface RT_Operation. 75
TABELA 4.3 - Atributos da da estrutura RT_Info. 76
TABELA 5.1 - Representação IDL da interface Task. 82
TABELA 5.2 - Representação IDL da interface PeriodicTask. 83
TABELA 5.3 - Representação IDL da interface AdaptiveService. 84
TABELA 6.1 - Configuração de *hardware* e *software* do cenário das experiências. 89
TABELA 6.2 - Alocação de tarefas entre os nodos da rede. 89
TABELA 6.3 - Média dos valores de período e atraso do sistema sem adaptação (i). 90
TABELA 6.4 - Média dos valores de período e atraso do sistema sem adaptação (ii). 91
TABELA 6.5 - Média dos valores de período e atraso do sistema com a primeira forma de adaptação (i). 92
TABELA 6.6 - Média dos valores de período e atraso do sistema com a primeira forma de adaptação (ii). 92
TABELA 6.7 - Médias de período e atraso para adaptação com SDY=-50000us e K_P=1/250000. 95
TABELA 6.8 - Relação de algumas das experiências realizadas. 95
TABELA 6.9 - Médias dos tempos na primeira rodada com a segunda versão de adaptação (i). 96
TABELA 6.10 - Médias dos tempos na primeira rodada com a segunda versão de adaptação (ii). 96
TABELA 6.11 - Tabela comparativa entre o sistema sem adaptação e com adaptação. 97

Resumo

CORBA vem se tornando o *middleware* padrão no desenvolvimento de aplicações distribuídas, tornando-as independentes de plataforma e linguagem. Ele tem sido utilizado também em aplicações de tempo real através de sua extensão para tempo real, o RT-CORBA. Apesar desta extensão ter conseguido reduzir vários dos problemas do CORBA no que se refere ao não-determinismo e falta de garantias temporais, ainda há muito estudo na área de mecanismos de escalonamento utilizados. Assim, este trabalho tem por objetivo apresentar uma proposta de escalonamento adaptativo no ambiente Real-Time CORBA. Nesta proposta o período das tarefas é controlado, variando dentro de uma faixa pré-estabelecida com o propósito de reduzir o atraso médio das tarefas da aplicação.

Palavras-chave: sistemas distribuídos de tempo real, escalonamento adaptativo, RT-CORBA, TAO.

TITLE: “AN APPROACH OF ADAPTIVE SCHEDULING IN THE REAL-TIME CORBA ENVIRONMENT.”

Abstract

CORBA is becoming the standard middleware in the development of distributed applications, which could become platform and language independent. It has been used in many real-time applications, through its extension for real-time systems, RT-CORBA. When it comes to CORBA's drawbacks such as no determinism and the lack of time and QoS guaranties, RT-CORBA has succeded in achieving satisfactory solutions. Besides, there is still a lot of research for new scheduling mecanisms. So, the aim of this work is to present an adaptive scheduling on Real-Time CORBA enviroment. In this proposal the task’s period is controlled, varying within a determined range, aiming to reduce the medium delay of the application’s tasks.

Keywords: real-time distributed systems, adaptive scheduling, RT-CORBA, TAO.

1 Introdução

O constante e acelerado desenvolvimento das redes de computadores permitiram o avanço na área da computação distribuída. Diversas alternativas de sistemas distribuídos, seja no âmbito de sistema operacional, seja como alternativas baseadas em *middleware* surgiram ao longo dos últimos anos. Ainda no final da década de 80 a OMG (*Object Management Group*) foi formada de modo a unificar as diversas soluções e teorias em um padrão aberto. Desta iniciativa surgiu o CORBA (*Common Object Request Broker Architecture*), um *middleware* que visa a diminuição das dificuldades existentes no desenvolvimento de aplicações em ambientes distribuídos heterogêneos, tanto em termos de sistema operacional como de linguagem de programação [MOW98] [ORF98]. Um grande leque de aplicações já está utilizando CORBA como a plataforma para sistemas distribuídos inclusive para a integração de sistemas legados [DAB2000] [FEL2000].

Sistemas de tempo real são aqueles cujo comportamento correto não depende apenas da integridade dos resultados obtidos (correção lógica), mas também dos valores de tempo em que são produzidos (correção temporal). Uma reação que ocorra além do prazo especificado pode ser sem utilidade ou até representar uma ameaça [FAR2000]. As aplicações de tempo real normalmente eram dominadas por tecnologias proprietárias e fechadas, porém, a partir da década de 80 a tecnologia de objetos distribuídos, e posteriormente de sistemas abertos se tornou cada vez mais popular na área de computação de tempo real, tendência que se tornou mais acelerada na década de 90 [SHO2000].

Apesar das especificações de objetos de serviços e facilidades comuns disponíveis no CORBA serem muito adequadas para aplicações distribuídas “normais”, os grandes requisitos temporais e de QoS (*Quality of Service*) das aplicações de tempo real tornam o CORBA deficiente e carente em vários aspectos como não-determinismo, falta de mecanismos para definir e garantir requisitos temporais e de QoS. Observando estas deficiências, a OMG partiu para a definição de uma extensão ao CORBA específica para aplicações de tempo real, a qual posteriormente recebeu a denominação de RT-CORBA (*Real-Time* CORBA). O RT-CORBA surgiu, portanto, para solucionar os problemas do CORBA para o desenvolvimento de aplicações distribuídas de tempo real [FEN2000] [ORY2000].

O RT-CORBA está chegando à sua segunda versão, mas ainda é um padrão em desenvolvimento. Foram criados interfaces e mecanismos para permitir a definição e a execução de uma grande variedade de aplicações de tempo real. Porém, muitos dos mecanismos que procuram manter a previsibilidade das aplicações dependem de outros aspectos como suporte do sistema operacional para garantir as exigências temporais, sistema de comunicação mais previsível e mecanismos e políticas de escalonamento que garantam a execução das tarefas no momento correto [GIL2000].

Nesse último aspecto é que residem muitas das questões em aberto existentes no RT-CORBA. Visto que as suas definições abordam apenas mecanismos de escalonamento, ainda há a necessidade de definições de políticas que, utilizando esses mecanismos, sejam capazes de se adequar à grande gama de classes de aplicações de tempo real existentes. Portanto, a escolha da política de escalonamento correta a uma determinada classe de aplicações é um dos pontos essenciais para o desenvolvimento de um sistema de tempo real. As diferenças de restrições entre essas diversas classes de

aplicações que possuem requisitos temporais exigem diferentes estratégias de escalonamento que melhor se aplicam a cada caso [GIL2000] [MON99].

Aplicações de *hard real time* não permitem perdas ou atrasos no que se refere a tempo. A correta execução de uma operação é obtida não apenas pela sua correta computação, mas também do tempo no qual ela é realizada. Aplicações dessa classe normalmente exigem um escalonamento em tempo de projeto, o qual possa garantir que o sistema irá respeitar os requisitos temporais necessários. Já aplicações de *soft real time*, assim como transmissões multimídia (áudio e vídeo) e jogos de computador permitem uma abordagem em tempo de execução e mesmo através de políticas de melhor esforço, onde detalhes podem ser excluídos para a manutenção do tempo da tarefa, ou mesmo pequenos atrasos são permitidos [OLI98]. Assim, observa-se uma grande gama de aplicações e, conseqüentemente, um conjunto ainda maior de soluções e políticas de escalonamento possíveis para utilização.

Muitas políticas de melhor esforço, em especial de técnicas adaptativas, estão presentes na literatura. Algumas das propostas encontradas se baseiam no conceito de realimentação, ou seja, tomam decisões sobre a atuação no sistema de acordo com informações coletadas do próprio sistema. As formas de atuação no sistema são variadas, abrangendo alterações em termos de flexibilização do *deadline*, do tempo de execução ou do período, por exemplo. Apesar do número expressivo de propostas encontradas na literatura ainda são poucas que se voltam para os sistemas distribuídos de tempo real abertos, mais especificamente o RT-CORBA.

Desta forma, focando-se nas aplicações de tempo real distribuídas compostas especialmente por tarefas periódicas, este trabalho tem por objetivo apresentar um modelo de escalonamento adaptativo para o ambiente RT-CORBA. A proposta apresentada é inspirada nos trabalhos que utilizam realimentação como forma de obtenção de informações para a tomada de decisões e naqueles que buscam na flexibilização dos períodos das tarefas a forma de aumentar o espectro de condições em que as aplicações executarão atendendo seus requisitos temporais.

Também é um dos objetivos deste trabalho mostrar um protótipo desenvolvido para validar o mecanismo apresentado. E, utilizando este protótipo, são feitos conjuntos de experiências que visam comparar o comportamento do sistema em um cenário experimental específico sem nenhuma forma de adaptação e com o mecanismo de adaptação proposto.

Este trabalho está organizado de modo que no capítulo 2 é feita uma breve revisão dos conceitos de sistemas e escalonamento de tempo real e um estudo de alguns mecanismos de adaptação encontradas na literatura. O capítulo 3 apresenta algumas características importantes do CORBA. No capítulo 4 são mostrados os motivos que levaram ao desenvolvimento do RT-CORBA e os detalhes mais importantes, bem como um estudo de caso sobre a implementação de ORB utilizada neste trabalho, o TAO (*The ACE ORB*). Finalmente, no capítulo 5 são apresentados o mecanismo de adaptação proposto neste trabalho e a descrição da implementação do protótipo sendo que os resultados das experiências são relatados no capítulo 6, o qual é seguido pelas conclusões e trabalhos futuros.

2 Mecanismos de Adaptação em Aplicações de Tempo Real

2.1 Caracterização de sistemas de tempo real

Uma grande parte dos sistemas computacionais atuais interagem permanentemente com os seus ambientes. Entre esses, distinguem-se os chamados *Sistemas Reativos* que reagem enviando respostas continuamente à estímulos de entrada vindos de seus ambientes. Sistemas de tempo real de uma forma geral se encaixam neste conceito de *Sistemas Reativos* [FAR2000]:

> *"Um sistema de tempo real é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos".*

A existência de prazos a serem cumpridos resulta em requisitos de natureza temporal sobre o comportamento desses sistemas. Portanto, o comportamento correto de um sistema de tempo real não depende somente da integridade dos resultados obtidos (correção lógica ou *correctness*), mas também dos valores de tempo em que são produzidos (correção temporal ou *timeliness*). Uma reação que ocorra além do prazo especificado pode ser sem utilidade ou até representar uma ameaça.

A maior parte das aplicações de tempo real se comporta então como sistemas reativos com restrições temporais. A concepção do sistema de tempo real é diretamente relacionada com o ambiente no qual está relacionado e com o comportamento temporal do mesmo. Nos sistemas embutidos e sistemas de supervisão e controle distinguem-se, além do próprio sistema computacional, o sistema a controlar e o operador, os quais correspondem ao ambiente do sistema computacional. Existem também situações nas quais as restrições temporais não são impostas pelo comportamento dinâmico de um eventual sistema a controlar, mas pelas exigências dos serviços a serem oferecidos a um usuário humano ou computacional. Nesses casos utiliza-se a noção de *Serviço de Tempo Real* como sendo um serviço que deve ser oferecido dentro de restrições de tempo impostas por exigências externas ao próprio sistema computacional. Então, generalizando, pode-se assumir [FAR2000]:

> *"Um Sistema de Tempo Real deve ser capaz de oferecer garantias de correção temporal para o fornecimento de todos os seus serviços que apresentem restrições temporais".*

Um dos equívocos mais comuns sobre sistemas de tempo real é de que para resolver todos os problemas com respeito às restrições temporais basta aumentar a velocidade computacional. A rapidez de cálculo visa melhorar o desempenho de um sistema, minimizando o tempo de resposta médio de um conjunto de tarefas. Ter um tempo de resposta curto não dá nenhuma garantia de que os requisitos temporais de cada processamento no sistema serão atendidos. Isto porque o desempenho médio não tem importância para o comportamento de um sistema composto de diversas atividades com restrições temporais. Muito mais importante do que a rapidez de cálculo, para os sistemas de tempo real, é o conceito de *previsibilidade* [FAR2000].

Um sistema de tempo real é dito *previsível* no domínio lógico e temporal quando, independentemente de variações de *hardware*, carga ou falhas, o comportamento do sistema pode ser antecipado antes de sua execução. De um ponto de vista rigoroso, para se assumir a *previsibilidade* de um sistema (ou de um serviço) de tempo real, precisa-se conhecer *a priori* o comportamento de um sistema, levando-se em conta a pior situação de carga (carga computacional de pico gerada pelo ambiente em um intervalo mínimo de tempo) ocorrendo simultaneamente com a hipótese de falhas (descreve os tipos e freqüências de falhas com os quais o sistema deve conviver).

Entretanto, a previsibilidade do sistema não depende apenas desses dois fatores (situação da carga e hipótese de falhas). Um conjunto de fatores ligados à arquitetura de *hardware*, ao sistema operacional e as linguagens de programação são também importantes. Ou seja, os tempos gastos na execução de códigos da aplicação (tempo de computação) no pior caso e em funções de suporte devem ser perfeitamente conhecidos ou determinados previamente.

O conceito de previsibilidade abordado até aqui pode ser denominado de "previsibilidade determinista". Na literatura também é encontrado o conceito de previsibilidade a uma antecipação probabilista do comportamento do sistema, baseada em estimativas ou simulações que estipulam probabilidades dos prazos a serem atendidos. A "previsibilidade probabilista" é útil em sistemas onde a carga computacional não pode ser conhecida *a priori* e portanto, não se consegue uma garantia em tempo de projeto do atendimento de todos os prazos.

Os sistemas de tempo real podem ser classificados a partir do ponto de vista de segurança em: (i) sistemas não críticos de tempo real (*soft real time systems*) quando as conseqüências de uma falha devidas ao tempo são da mesma ordem de grandeza que os benefícios do sistema em operação normal (tais como jogos, sistemas multimídia, sistemas bancários, entre outros) e (ii) sistemas críticos de tempo real (*hard real time systems*) quando as conseqüências de pelo menos uma falha temporal excedem em muito os benefícios normais do sistema (por exemplo, sistemas de controle de vôo, sistemas de controle de plantas nucleares, etc.). Previsibilidade probabilista e determinista são associadas aos dois grupos acima, respectivamente.

A classificação dos sistemas de tempo real ainda pode ser feita baseando-se no ponto de vista da implementação em dois tipos: (i) sistemas de resposta garantida (*guaranteed response system*) onde se sabe em tempo de projeto que existem recursos suficientes para suportar a carga de pico e o cenário de falhas definido; e (ii) sistemas de melhor esforço (*best effort system*) quando a estratégia de alocação dinâmica de recursos se baseia em estudos probabilistas sobre a carga esperada e os cenários de falhas aceitáveis.

A maior parte dos sistemas de tempo real vem sendo projetada de acordo com o paradigma de melhor esforço, satisfatório de fato apenas para os sistemas não críticos de tempo real e levando a situações danosas para os sistemas críticos para os quais é aconselhado seguir o paradigma de resposta garantida.

2.2 Classificação das abordagens de escalonamento de tempo real

As aplicações de tempo real, para efeito de escalonamento, podem ser expressas pelo conjunto de tarefas que as compõem, pela carga computacional que estas tarefas determinam, bem como pelo conhecimento que se tem desta carga. Quando a carga computacional apresenta característica **estática** ou **limitada** se tem o conhecimento em

tempo de projeto das informações sobre cada tarefa, como suas restrições temporais, seus tempos de chegada, execução e até mesmo as possíveis situações de pico. As cargas estáticas normalmente são modeladas através de tarefas periódicas e esporádicas. No caso oposto, quando não se tem o conhecimento prévio das características das tarefas, a carga é denominada de **dinâmica** ou **ilimitada**. Neste caso, as cargas são modeladas utilizando-se tarefas aperiódicas [FAR2000].

Escalonamento é definido como o processo de ordenar as tarefas na fila de pronto, sendo que o escalonador é o componente do sistema responsável por implementar uma política de escalonamento a fim de ordenar o conjunto de tarefas para execução no processador. O escalonamento é geralmente dividido em duas etapas: o teste de *escalonabilidade* e o cálculo da escala de execução. O teste de *escalonabilidade* é executado inicialmente para determinar se as restrições temporais do conjunto de tarefas em questão podem ser alcançadas de acordo com a abordagem de ordenação definida na política de escalonamento [FAR2000].

Várias propostas de políticas de escalonamento podem ser encontradas na literatura, cada uma apresentando diferenças significativas. Considerando aspectos relevantes como a previsibilidade e garantias oferecidas, a utilização de recursos e os algoritmos empregados, a FIGURA 2.1 mostra uma das taxonomias possíveis para o conjunto de propostas da literatura [FAR2000] [MON99] [OLI97].

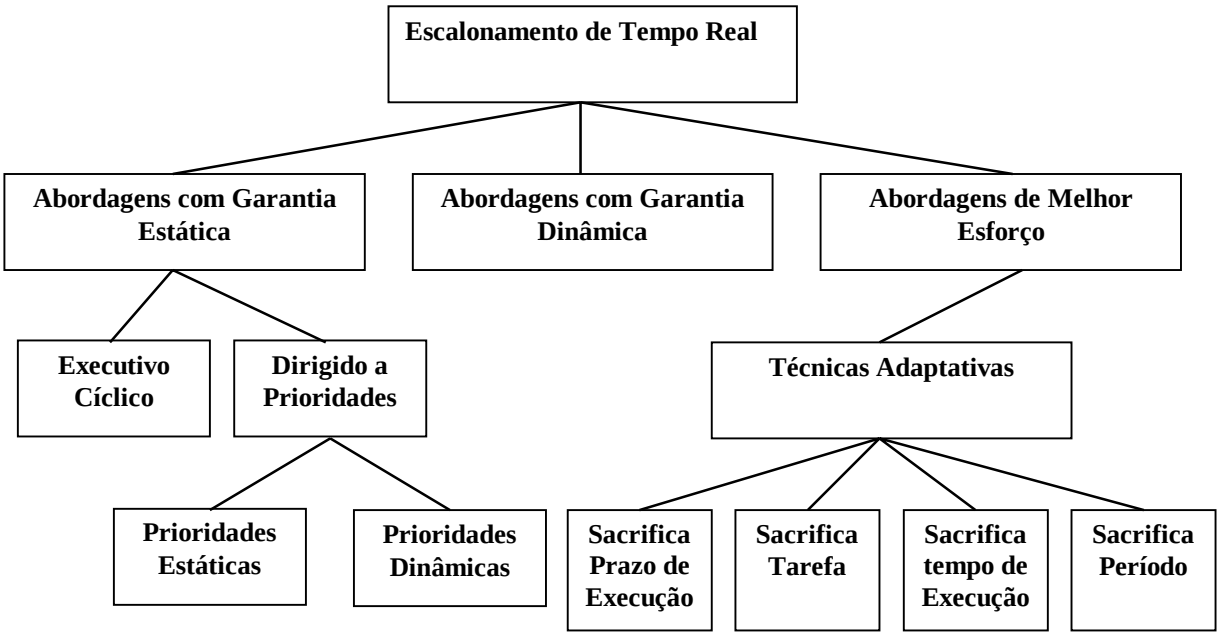


FIGURA 2.1 - Classificação das Abordagens para Escalonamento de Tempo Real.

Na abordagem de **executivo cíclico** todo o processo de teste de *escalonabilidade* e o trabalho de escalonamento são realizados em tempo de projeto ("*offline*"). O resultado é uma grade ("*time-grid*") que determina qual tarefa roda quando e em qual processador.

Nos **algoritmos baseados em prioridade** é feito também o teste de *escalonabilidade* em tempo de projeto de forma a determinar se todas as tarefas podem ser executadas dentro dos *deadlines* e atribuir as prioridades conforme a política escolhida. Em tempo de execução um escalonador preemptivo executa as tarefas habilitadas conforme as suas prioridades. As propostas mais significativas da literatura

tratam com prioridades fixas, mas também existem alternativas baseadas em prioridades variáveis.

As abordagens de escalonamento estáticas têm a vantagem de oferecer previsibilidade determinista para o cumprimento dos prazos. Claro que para conseguir isso se deve sempre considerar o pior caso de tempo de execução das tarefas, o que pode causar subutilização de recursos. Outra limitação desta abordagem é a exigência de carga limitada e estática [OLI97].

No caso das abordagens com **garantia dinâmica** são utilizados testes de aceitação, baseados em análises realizadas com hipóteses de pior caso sobre alguns parâmetros temporais para verificar a *escalonabilidade* do conjunto formado por uma nova tarefa que chega no sistema e as tarefas que já existiam previamente na fila de pronto. Caso o teste indique que o novo conjunto não é escalonável, a tarefa é rejeitada, preservando assim o conjunto de tarefas que já eram escalonáveis. Essas abordagens são próprias para aplicações que possuam restrições críticas, mas que operam em ambientes que não são deterministas [FAR2000].

No escalonamento baseado em **melhor esforço** não há garantias em tempo de projeto de que os *deadlines* serão cumpridos. Os sistemas apenas fazem o possível para que os prazos sejam cumpridos. A utilização de sistemas de melhor esforço tem como conseqüência a possibilidade de sobrecargas quando não é possível que todas as tarefas cumpram seus respectivos prazos. Assim, é necessário que se trate com esta sobrecarga através de: descarte de algumas tarefas; execução de todas as tarefas, mas com sacrifício no prazo de execução de algumas; execução de todas as tarefas, mas com sacrifício no tempo de execução de algumas ou execução de todas as tarefas, mas sacrificando o período de algumas tarefas periódicas.

Para o tratamento da sobrecarga é necessário definir qual a forma de sacrifício de tarefas que será utilizado, bem como quais as tarefas que serão sacrificadas. O mecanismo é executado para minimizar uma função de erro ou para maximizar uma função de benefício do sistema.

2.3 Cenários para adaptação de aplicações de tempo real

Mesmo em sistemas computacionais construídos especificamente para a execução de aplicações de tempo real existem amplas oportunidades para adaptação. Isto pode ocorrer porque a carga representada pela aplicação em questão não possui um limite conhecido que possa ser garantido em projeto, ou porque, mesmo que a carga possua uma demanda por recursos bem caracterizada, pode não ser economicamente viável garantir o seu comportamento em um cenário de pior caso.

Embora a maior parte da literatura de tempo real trate de adaptação através da negociação da qualidade de serviço, a adaptação também pode ocorrer através de uma ação unilateral. No caso de uma adaptação unilateral existem os seguintes cenários possíveis [OLI98]:

- Uma variação no comportamento da aplicação gera como resposta uma adaptação do suporte. Por exemplo, a redução no período de uma tarefa da aplicação é detectada pelo suporte que, em resposta, aumenta o percentual do tempo de processador reservado para esta tarefa.

- Uma variação no comportamento do suporte gera como resposta uma adaptação do próprio suporte. Por exemplo, em um sistema distribuído a falha de um processador pode ser compensada pela redistribuição da carga.
- Uma variação na aplicação gera como resposta uma adaptação da própria aplicação. Em sistemas onde a aplicação pode reservar recursos, cabe à própria aplicação administrar os recursos reservados. Uma alteração no ambiente poderá exigir uma alteração no modo de operação (*mode change*) da aplicação e uma redistribuição interna dos recursos previamente reservados. Por exemplo, em Beccari et al. [BEC99] uma sobrecarga na utilização do processador causa alteração no período das tarefas de *soft real time* (será detalhado na seção 2.5).
- Uma variação no comportamento do suporte gera como resposta uma adaptação da aplicação. Esta situação é muito comum em sistemas distribuídos, onde variações no atraso podem estar associadas com o envio de mensagens pela rede. A diferença entre os processadores que compõem a rede também pode ser causa de alterações nos atrasos. Normalmente essas variações devem ser contornadas pela aplicação visto que os suportes existentes não são capazes de isolar a aplicação.

2.4 Mecanismos de adaptação em aplicações de tempo real

Aplicações de tempo real em ambiente distribuído podem estar sujeitas a variações nos tempos de resposta das tarefas. Essas variações podem ser causadas por atrasos no envio de mensagens pela rede ou na alteração de algum nodo do qual a aplicação também é dependente. Isto pode acontecer ao longo do tempo de execução da aplicação, e não apenas em sua etapa de inicialização. Logo, existe a necessidade de uma adaptação contínua durante a execução da aplicação.

No contexto de sistemas distribuídos, a adaptação é um processo complexo que envolve um conjunto de componentes e sistemas. Apesar da multiplicidade de abordagens, o objetivo básico da adaptação pode ser considerado sempre o mesmo: estender a amplitude de condições nas quais uma aplicação pode ser executada corretamente [GEC97].

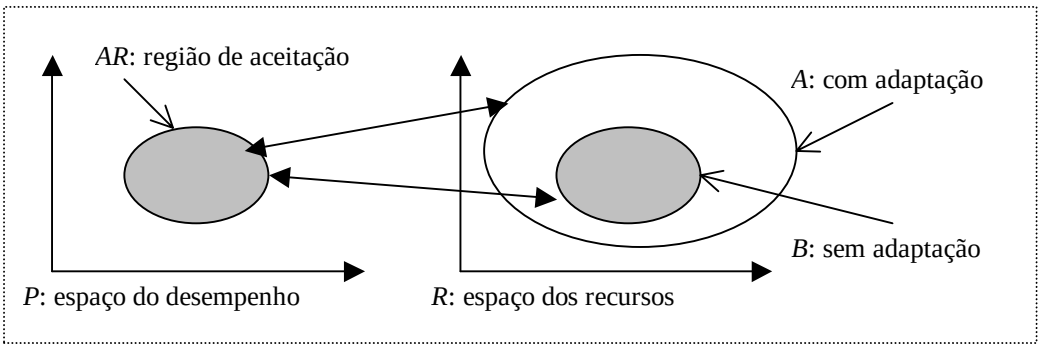


FIGURA 2.2 - Efeito da adaptação na área de aceitação no espaço de recursos.

Na FIGURA 2.2 [GEC97] a adaptação pode ser visualizada pela definição de dois espaços: o espaço de desempenho (*P*), e o espaço de recursos (*R*). As dimensões em *P* incluem medidas de QoS definidas pelo usuário, tempo de resposta, funcionalidade e

outros fatores importantes relativos ao desempenho da aplicação. A região *AR* (*acceptance region*) é a área de *P* na qual considera-se que a aplicação está executando apropriadamente. As dimensões em *R* correspondem às características dos recursos no ambiente de operação, tais como memória, largura de banda, taxa de perdas e ocupação de processador entre outras.

Supondo uma dada aplicação sem adaptação pode-se definir um mapeamento entre *R* e *P* que mapeia *AR* na região *B* em *R*. Introduzindo a adaptação, o mapeamento é alterado de modo que *AR* passa agora a ser mapeado para a região maior *A*. Normalmente, espera-se que *A* inclua *B* e que, portanto, a eficiência da adaptação possa ser representada como sendo o resultado da operação de (*A*-*B*).

Dinamicamente, a adaptação trabalha com um conjunto de mecanismos cujo objetivo é manter o ponto de operação dentro da região *AR*. Quando este ponto de operação se move para fora desta região, um controlador inicia uma ação de correção (a adaptação) para trazer o ponto de operação de volta ao interior da região de aceitação. Usualmente, o controlador faz isto através da observação de certos parâmetros do sistema e então executa o algoritmo de adaptação. A ação resulta na alteração dos parâmetros observados pelo controlador, como pode ser visto na FIGURA 2.3 [GEC97].

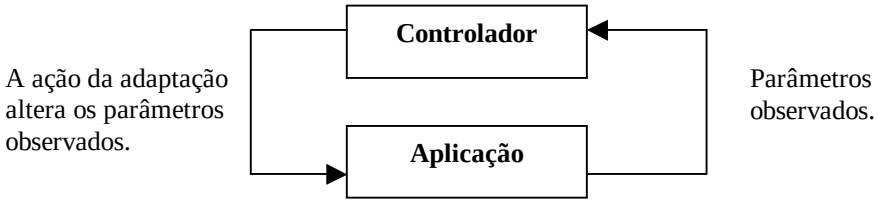


FIGURA 2.3 - Mecanismo básico de adaptação.

Considerando estes princípios básicos da adaptação, no restante desta seção serão vistas algumas técnicas de adaptação descritas na literatura [OLI99] [OLI98].

Flexibilização do deadline

O conceito de *deadline* é importante principalmente por duas razões:

- Uma aplicação pode usar os *deadlines* de suas tarefas para definir as prioridades dos seus fluxos de execução. Embora os sistemas operacionais em geral não suportem o conceito de *deadline*, a grande maioria deles suporta o conceito de prioridades. Desta forma, a aplicação pode usar os *deadlines* das tarefas como ponto de partida para a definição das respectivas prioridades. Na literatura, as políticas mais conhecidas neste sentido são o EDF (*Earliest Deadline First* [LIU73]), o Taxa Monotônica (*Rate Monotonic* [LIU73]) e o *Deadline* Monotônico (*Deadline Monotonic* [AUD93]).
- A aplicação pode comparar os *deadlines* das tarefas com o tempo de resposta efetivamente observado e assim, é possível obter uma quantificação do desempenho da aplicação com respeito aos seus requisitos temporais. Podem ser citadas duas formas básicas de quantificações: (i) o somatório dos atrasos de todas as tarefas e (ii) a taxa de tarefas que perderam o respectivo *deadline*. A primeira fornece uma medida do desempenho mais apropriada quando as tarefas devem ser executadas mesmo com atraso. A segunda é uma medida mais útil quando as tarefas possuem *deadline* firme (*firm deadline*), isto é, não

há benefício em executar uma tarefa após o seu *deadline*. Uma proposta, variação do *deadline* firme, é a *deadline* (m,k)-*firm* descrita por Hamdaoui & Ramanathan [HAM97], na qual em cada tarefa periódica devem ser concluídas dentro do *deadline* pelo menos *m* ativações em qualquer conjunto de *k* ativações consecutivas. Neste modelo, a taxa de *deadlines* perdidos deve ser adaptada para considerar os valores de *m* e *k* associados a cada tarefa.

A forma mais simples e freqüente de adaptação de flexibilização do *deadline* é simplesmente relaxar o próprio conceito de *deadline*. Exemplificando, Jensen et al. [JEN85] apresenta uma abordagem em que a conclusão de cada tarefa contribui para o sistema com um benefício, e o valor deste benefício pode ser expresso em função do instante de conclusão da tarefa (*time value function*).

Nesse mecanismo, a adaptação ocorre de forma implícita no momento em que os *deadlines* não são respeitados e as tarefas são concluídas com atraso. Esta forma de adaptação tem como vantagem a facilidade, visto que os projetistas e programadores não precisam incluir código para implementar o mecanismo. Entretanto, mesmo que os aspectos teóricos indiquem que é possível a aceitação dos atrasos, o conhecimento dos *deadlines* e de suas respectivas importâncias permite uma degradação mais suave do que quando *deadlines* são perdidos de forma aleatória.

Flexibilização do tempo de execução

Neste mecanismo de adaptação, as tarefas são escalonadas de forma a cumprirem seus *deadlines*. Em caso de sobrecarga, o tempo de execução da tarefa é reduzido. Esta abordagem pode ser resumida como "fazer o trabalho possível dentro do tempo disponível" e na literatura é denominada de Computação Imprecisa (*Imprecise Computation* [LIU94]).

A Computação Imprecisa está fundamentada na idéia de que cada tarefa pode ser dividida em uma parte obrigatória (*mandatory*) e uma parte opcional (*optional*). A parte obrigatória é capaz de gerar um resultado, chamado de impreciso (*imprecise result*), com a qualidade mínima aceitável. A parte opcional, por sua vez, refina o resultado gerado pela parte obrigatória, até que ele alcance a qualidade desejada, esse resultado é chamado de preciso (*precise result*). Uma tarefa é chamada de "tarefa imprecisa" (*imprecise task*) quando é possível decompô-la nas partes obrigatória e opcional. Algumas tarefas podem possuir apenas a parte opcional e outras, apenas a parte obrigatória, porém, a técnica não exige que cada tarefa possua as duas partes.

Em situações normais, tanto a parte obrigatória quanto a opcional são executadas e a qualidade máxima é obtida. Em situações de sobrecarga algumas partes opcionais são deixadas de lado. Consegue-se uma degradação controlada do sistema, na medida em que se pode determinar o que não será executado em caso de sobrecarga. Na Computação Imprecisa o tempo de computação é negociado a partir da introdução do conceito de qualidade do resultado. A redução da qualidade do resultado produzido gera uma redução na demanda pelos recursos do sistema, o que permite o melhor atendimento dos *deadlines*.

Existem pelo menos três formas básicas de programar tarefas imprecisas:

- **Funções monotônicas (*monotone functions*)**: são aquelas cuja qualidade do resultado aumenta (ou pelo menos não diminui) na medida em que o tempo de execução da função aumenta. As computações necessárias para obter-se um

nível mínimo de qualidade correspondem à parte obrigatória. Qualquer computação além desta, refina progressivamente o resultado e é tratada como parte opcional.

- **Etapas de melhoramento *(sieve functions)***: são aquelas cuja finalidade é produzir saídas no mínimo tão precisas quanto as correspondentes entradas. Tipicamente, não existe benefício em executar uma etapa de melhoramento parcialmente.
- **Múltiplas versões *(multiple versions)***: normalmente são empregadas duas versões. A versão primária gera um resultado preciso, porém com tempo de execução maior. A versão secundária tem um tempo de execução menor, porém gera um resultado impreciso.

A forma de programação empregada interfere no tipo de decisão de escalonamento a ser tomada. Quando uma função monotônica é empregada qualquer tempo a mais do processador fornecido para a tarefa resulta em uma (pequena) melhora na qualidade do resultado. Entretanto, quando se usa etapas de melhoramento (nas quais não há vantagem de executar parcialmente uma etapa) ou múltiplas versões (onde normalmente existem apenas duas versões) a decisão a ser tomada é do tipo "sim ou não", ou seja, ou se executa a parte opcional ou não.

Flexibilização do período

Em uma aplicação de tempo real muitas tarefas são executadas periodicamente. Em geral estas tarefas possuem o seu período definido em tempo de projeto. Uma forma de prover adaptabilidade à aplicação é permitir que este período possa variar dinamicamente, durante a execução da aplicação. Desta forma, a qualidade da aplicação, representada aqui pelo período das tarefas, seria adaptada ao desempenho do suporte onde estiver executando.

Por exemplo, a taxa de exibição dos quadros em um vídeo (*frame rate*) pode ser alterada conforme o desempenho do suporte onde a aplicação executa. Uma transmissão de áudio em tempo real é capaz de apresentar melhor qualidade quando pedaços menores são transmitidos com maior freqüência. Há propostas na literatura que lidam com a escolha dos períodos das tarefas em função do desempenho do sistema em tempo de projeto (*offline*). Outros trabalhos analisam a situação onde os períodos das tarefas podem ser alterados em tempo de execução (*online*).

Flexibilização da execução

Uma forma mais radical de flexibilização é simplesmente não executar algumas tarefas quando o desempenho estiver abaixo do desejado. Este mecanismo pode ser considerado um caso extremo dos anteriores: (i) quando o *deadline* é relaxado a tal ponto que a tarefa somente será concluída em um tempo infinito, (ii) quando a qualidade é sacrificada a ponto do tempo de execução da tarefa chegar a zero ou, ainda, (iii) quando a tarefa passa a ser executada com período infinito.

Flexibilização do nível de paralelismo

Muitas aplicações dividem o trabalho a ser feito em partes que podem ser executadas em paralelo com o objetivo de diminuir o tempo de resposta da tarefa. Caso a arquitetura usada não suporte execução em paralelo, as partes são executadas seqüencialmente. O resultado é o mesmo, apenas o tempo de execução será maior. Esta organização do algoritmo permite sua adaptação a diferentes níveis de paralelismo, ou seja, é capaz de se adaptar para tirar proveito do paralelismo real quando este existir.

Atualmente, ainda são poucos os sistemas que suportam paralelismo real. Entretanto, a tendência é que este número cresça. Computadores com dois ou quatro processadores já são encontrados com certa freqüência. Na medida em que arquiteturas paralelas tornam-se mais comuns, este mecanismo de adaptação poderá ser utilizado para tirar proveito do paralelismo existente, sem impedir a execução da aplicação em máquinas monoprocessadas.

2.5 Propostas de Adaptação na Literatura

Na literatura são encontradas várias propostas de como medir a qualidade do sistema e de como atuar no mesmo buscando o benefício dessa medida. Por exemplo, em Lu et al. [LU2000] a qualidade do sistema é medida através da taxa de perda de *deadline* das tarefas e da taxa de utilização do sistema em uma janela de medição. A atuação é, então, feita de modo a alterar o modo de operação das tarefas e na decisão de aceitar ou rejeitar tarefas para execução.

Em outra abordagem, Brandt et al. [BRA98] apresenta uma abordagem de adaptação que procura suportar aplicações de *soft real time* em sistemas operacionais de propósito geral. Nesta proposta, cada aplicação especifica um conjunto de modos (algoritmos) nos quais ela pode executar, além das exigências computacionais e o correspondente benefício de ser executada em cada modo. Assim, cada aplicação é caracterizada pela sua máxima utilização de processador e pelo seu máximo benefício. Cada nível que a aplicação define é caracterizado pela sua relativa utilização de processador, pelo benefício relativo e o seu período. A medição da utilização do processador durante a execução das aplicações determina qual dos níveis de execução que será utilizado para cada aplicação. A proposta apresenta uma implementação baseada em um *middleware* denominado de *Dynamic QoS Resource Manager* (DQM) que recebe os dados estimados para cada aplicação e atua alterando os níveis de execução conforme o resultado da medição da utilização de processador do processo *idle* do sistema operacional através de um mecanismo específico encontrado em sistemas operacionais UNIX e Linux.

Pensando nas condições de operação de robôs autônomos (monoprocessados) que precisam se adaptar a alterações repentinas na sua carga de tarefas de tempo real e geralmente a situações de sobrecarga, Beccari et al. [BEC99] apresenta uma técnica de modulação de freqüência de um conjunto de tarefas periódicas. Esta técnica inicialmente separa as tarefas periódicas que compõem o sistema em dois conjuntos: as de *soft real time* e as de *hard real time*. Sobrecargas no sistema são detectadas através da medição da taxa de utilização do mesmo e então, baseando-se em uma política de *graceful degradation* a parcela de utilização relativa ao conjunto das tarefas de *soft real time* é relaxada de modo a controlar as condições de sobrecarga. No artigo são

apresentadas algumas alternativas de algoritmos para modulação da freqüência, sendo que no primeiro deles é utilizada uma alteração linear nos períodos das tarefas e em outro se procura preservar a prioridade das tarefas de modo que tarefas de mais baixa prioridade são saturadas (é escolhido o período máximo) e em outras (prioridade intermediária) é aplicado um fator de modulação.

Em Shin & Meissner [SHI99], por sua vez, é apresentada uma proposta de adaptação e *graceful degradation* em sistemas de tempo real multiprocessados. No artigo é apresentada uma proposta para realocação da utilização do processador através de mudanças nos períodos das tarefas em ambiente multiprocessado de modo a lidar com mudanças na prioridade das tarefas, tais como as advindas de troca de modo de operação, e para recuperação em caso de falhas. A adaptação é feita através de pequenos incrementos ou decrementos nos períodos das tarefas de modo a otimizar o desempenho do sistema oferecendo maior utilização do processador para tarefas mais importantes. Cada tarefa possui um valor de importância associado às suas diferentes freqüências, e o objetivo da adaptação também é maximizar a soma dos valores de importância das tarefas que compõem o sistema. O artigo apresenta ainda, uma proposta para realocação de tarefas em caso de falhas nos processadores. O objeto da realocação é, em primeiro lugar, gerar uma alocação com um alto valor de importância e, em segundo lugar, minimizar o número de tarefas movimentadas. São apresentados dois algoritmos que mudam a utilização das tarefas em incrementos discretos, alterando de uma freqüência permitida para outra e, finalmente, associando as tarefas aos processadores. A validação do mecanismo proposto foi feita através da simulação do sistema, utilizando um conjunto randômico de tarefas e em uma segunda experiência, uma aplicação exemplo.

Outros trabalhos encontrados na literatura que seguem pela mesma direção podem ser encontrados em Welch et al. [WEL98] e Abdelzaher et al. [ABD97] [ABD98]. A TABELA 2.1 apresenta uma comparação qualitativa de algumas propostas de abordagens adaptativas encontradas na literatura citadas anteriormente.

TABELA 2.1 - Análise qualitativa de algumas propostas de técnicas adaptativas da literatura.

Proposta	Medição	Atuação	Experiências	Plataforma
Lu et al. [LU2000]	Taxa de perda de deadline e taxa de utilização do sistema em uma janela de medição.	Alteração do modo de execução das tarefas e na decisão de aceitar ou rejeitar novas tarefas.	Simulação	Monoprocessador
Brandt et al. [BRA98]	Utilização do processador.	A utilização do processador determina o modo de operação do sistema (requisitos e benefício relativo)	Sistemas Linux/UNIX	Monoprocessador
Beccari et al. [BEC99]	Taxa de utilização do sistema.	Alteração do período das tarefas soft real time (não há alteração nas de hard real time), graceful degradation.	Simulação e protótipo ainda incompleto para um sistema VME-based.	Monoprocessador
Shin & Meissner [SHI99]	Qualidade do sistema (cada tarefa possui um benefício dependendo do período).	Alteração do período das tarefas e/ou realocação de tarefas.	Simulação (tarefas randômicas e aplicação exemplo)	Multiprocessador
Abdelzaher et al. [ABD98]	Throughput em pacotes por segundo.	Alteração no nível de QoS (comunicação) exigido pelas tarefas.	The Open Group (TOG) 7.2 kernel	Cliente/Servidor (sistema de comunicação)

Abdelzager et al. [ABD97]	Na chegada de uma tarefa é calculado o benefício do sistema com essa nova tarefa.	Aceita ou rejeita a execução dos módulos (partes que compõem uma tarefa), ou a tarefa inteira, de acordo com o nível de QoS negociado.	RTPOOL implementado sobre OSF Mach RT-mk7.2	Multiprocessador

2.6 Considerações Finais

Sistemas de tempo real são aqueles que exigem, além da correção lógica, a correção temporal de todos os seus serviços oferecidos. Ao contrário do que muitos pensam, apresentar um tempo de resposta curto não garante que os requisitos temporais serão respeitados. Isto se deve ao fato de que o desempenho médio não tem importância para o comportamento de um sistema composto por atividades com restrições temporais. Para sistemas de tempo real o mais importante é a previsibilidade.

Os sistemas de tempo real podem ser classificados em: (i) *hard real time*, quando uma falha temporal pode ter conseqüências mais prejudiciais do que o benefício que o funcionamento normal do sistema possui e (ii) *soft real time*, quando uma falha temporal tem conseqüência na mesma ordem de grandeza que o benefício trazido pelo funcionamento normal do sistema. Ainda, pode-se apresentar uma classificação baseada na implementação dos sistema em (i) sistemas de resposta garantida e (ii) sistemas de melhor esforço.

O escalonamento, que é o processo de ordenar as tarefas na fila de pronto, é muito importante para os sistemas de tempo real. Neste capítulo foram abordadas diversas políticas de escalonamento, desde àquelas que lidam com cargas estáticas e dão garantias deterministas, até aquelas que conseguem lidar com carga dinâmica proporcionando garantias probabilistas, bem como uma taxonomia possível para este conjunto de políticas.

Mais recentemente, a utilização de políticas de escalonamento de melhor esforço e técnicas adaptativas têm aumentado, seja porque a carga de certas aplicações não pode ser determinada em tempo de projeto ou porque, mesmo que essa carga seja conhecida, não é economicamente viável garantir os recursos necessários para garantir seu comportamento no cenário de pior caso.

Neste capítulo foram apresentadas algumas abordagens para técnicas adaptativas encontradas na literatura, que podem ser enquadradas nos seguintes tipos básicos: flexibilização do *deadline*, flexibilização do tempo de execução, flexibilização do período, flexibilização da execução e flexibilização do nível de paralelismo. Apesar da gama de propostas encontradas, o mecanismo básico pode ser considerado sempre o mesmo: estender a amplitude de condições nas quais uma aplicação pode ser executada corretamente.

3 CORBA

Impulsionadas pelo avanço das redes de computadores e da computação distribuídas várias arquiteturas que visavam facilitar a criação de aplicações baseadas em objetos distribuídos foram propostas. Entre as mais conhecidas encontram-se: Java/RMI, Jini (Sun), *Microsoft* DCOM (*Distributed Component Object Model*), mais recentemente *Microsoft* .NET, e finalmente CORBA (*Component Object Request Broker Architecture*).

Criado por um consórcio de mais de 800 empresas, o CORBA é uma das alternativas mais completas e ambiciosas das que surgiram, trazendo como promessa a independência de plataforma e de linguagem. É baseado em componentes (objetos) que podem descobrir um ao outro e se intercomunicar através de um barramento de objetos (um *middleware*), além de oferecer vários outros serviços.

Muitos dos benefícios trazidos pelo CORBA residem na utilização de uma linguagem independente (IDL – *Interface Definition Language*) para a definição da interface dos objetos. Com a existência de mapeamentos de IDL para várias outras linguagens utilizadas no mercado, pode-se incorporar aos sistemas distribuídos até mesmo aplicações legadas e outros softwares não escritos originalmente para estes sistemas [DAB2000].

Este capítulo tem por objetivo apresentar uma breve visão das características do CORBA bem como os elementos fundamentais de sua arquitetura. Também serão detalhados alguns dos serviços definidos como, por exemplo, o serviço de nomes e o de eventos.

3.1 OMG – Object Management Group

Criada por volta de 1989 por um conjunto de algumas poucas empresas, a OMG (*Object Management Group*) tem por objetivo coordenar o desenvolvimento de padrões na área de objetos distribuídos. Supunha-se na época, em um momento ainda inicial do desenvolvimento de objetos distribuídos, que a falta de um órgão de mediação levaria as várias frentes existentes a criar a sua própria arquitetura para sistemas distribuídos, tornando a interoperabilidade impossível.

A OMG é, portanto, um consórcio internacional de empresas que não visa lucro, tendo por objetivo definir padrões na área de computação com objetos distribuídos [OMG2000]. Cabe ressaltar que a OMG não é um órgão oficial de padronização. Da mesma forma, não faz parte de órgãos oficiais como a ISO (*International Standards Organization*) e ANSI (*American National Standards Institute*) [POP98].

Ela é composta por um conjunto de membros organizados em diferentes grupos, sendo que cada grupo possui direitos em relação à participação no processo de definição de padrões e nas votações durante esse processo. A inclusão de um membro em um dos grupos é feita de forma livre sendo que, a princípio, qualquer empresa, órgão de governo, ou universidade, interessada na área de objetos distribuídos, seja para participar do processo de criação dos padrões, seja para apenas acompanhar as versões mais atuais desses padrões pode se inscrever na organização. Na TABELA 3.1 podem ser vistos alguns dos grupos de membros, sua definição e alguns dos principais membros [OMG2000a].

TABELA 3.1 - Grupos de membros da OMG.

Grupo	Definição	Principais Membros
Contributing Members	Possuem direito de voto tanto no DTC (*Domain Technical Committee*) como no PTC (*Plataform Technical Committee*).	AOL, AT&T, Compaq, Computer Associates, Ericsson, Hewlett-Packard, Informix, Inprise Corporation, Lucent Technologies, Oracle, Sun Microsystems, W3 Consortium, Xerox
Domain Members	Representam organizações ou indústrias que implementam e disponibilizam produtos utilizando a tecnologia de objetos.	Los Alamos National Laboratory, Boeing
Influencing Members	São participantes importantes na organização, que possuem direito de voto nos grupos de interesse especial e nas forças tarefa.	CERN (Organização Européia para Pesquisa Nuclear), Cisco Systems, Novell
Government Members	São entidades ligadas a governos e possuem os mesmos direitos dos *Influencing Members*	Defense Information Systems Agency (Estados Unidos), NASA Ames Research Center Aviation Safety Program
Auditing Members	Podem comparecer a todas as reuniões, mas com número de participantes limitado a uma pessoa por empresa e sem direito de voto.	Microsoft
University Members	É formado por universidades de todo o mundo, os representantes podem comparecer a todas as reuniões, tendo direito de voto nos grupos de interesse especial e nas forças tarefa.	Colorado State Univ, Dept of Computer Science, Lancaster University, University of Frankfurt, Universidade de São Paulo (Escola Politécnica)

O objetivo da OMG não é criar produtos, mas por outro lado, é sua constante preocupação a criação de especificações que possam ser implementadas por qualquer empresa que o deseje [MOW98]. O desenvolvimento dessas especificações passa por várias etapas começando pela consulta à comunidade (começando pelos membros da OMG) sobre novas propostas ou extensões de padrões já existentes. Com o lançamento das RFPs (*Request For Proposal*) vários grupos de empresas podem desenvolver suas propostas.

Depois das rodadas de votações a proposta que melhor atenda as necessidades da comunidade é escolhida. Essa proposta pode ser uma das concorrentes ou uma união das suas melhores características. Cabe ressaltar que as decisões tomadas pelos membros da OMG são baseadas em implementações já existentes e não em tecnologias que se encontram apenas no papel [POP98].

Assim, muitas vezes as decisões tomadas pela OMG são baseadas em premissas de marketing e não técnicas, em grande parte porque as submissões, em resposta a uma RFP, são baseadas em produtos comerciais que tem uma certa demanda de mercado.

Sob alguns aspectos pode-se criticar a lentidão na criação de padrões, ou até mesmo a precipitação, ou ainda, a atitude mercadológica buscando a atração de um maior número de membros. Entretanto, são louváveis os resultados obtidos pela OMG na obtenção de consenso entre a comunidade. Ainda que algumas das escolhas tomadas não sejam as melhores em termos técnicos, elas são escolhas baseadas em elementos de tecnologia que realmente podem ser implementados por todos.

Desta forma a OMG conseguiu trazer uma certa organização na área de objetos distribuídos sem dominar e, portanto, limitar a criatividade dos desenvolvedores da área. Isso pode ser comprovado pela existência de outras alternativas ao CORBA encontradas no mercado como, por exemplo, a tecnologia DCOM e mais atualmente a tecnologia .NET da *Microsoft*.

3.2 OMA – Object Management Architecture

A OMA *(Object Management Architecture*) é definida como um modelo de referência para a tecnologia de objetos. É importante, inicialmente, identificar alguns pontos relevantes no que se refere às intenções da OMG ao definir a OMA [POP98]:

- O modelo de objetos definido não visa ser um meta-modelo. Não há a intenção de criar um modelo flexível e abrangente que possa ser capaz de se tornar um modelo dos modelos existentes. Ele é, pelo contrário, um modelo específico à OMG.
- Também não é um super conjunto das implementações existentes, e essas não necessitam se enquadrar perfeitamente ao modelo, e ainda, podem solicitar alterações na OMA existente visto que essa não é estanque.
- Ainda, o modelo não é baseado em idéias puramente teóricas. O objetivo é implementar o modelo guiando a tecnologia existente para chegar a esta meta.

O desenvolvimento dessa arquitetura seguiu certos objetivos técnicos para que trouxesse benefícios ao gerenciamento de objetos, os quais podem ser listados abaixo [POP98]:

- **Conformidade**: deve ser mantida a compatibilidade na herança de interfaces, onde as interfaces derivadas devem possuir pelo menos a interface da superclasse; herança de implementação, onde a chamada de um método de uma classe derivada implica na execução do código da classe pai; programa ou processo e objetos.
- **Transparência de distribuição**: tem como objetivo tornar transparente para o programador todos os aspectos referentes à distribuição de objetos tais como: localização, forma de acesso, mecanismos de comunicação, meio de armazenamento, tipos de máquinas e de sistemas operacionais, segurança, entre outros.
- **Desempenho tanto para operações remotas quanto locais**: deve apresentar desempenho e escalabilidade (no que se refere ao aumento do número de clientes de um certo objeto) tanto no acesso local quanto no acesso remoto.
- **Comportamento extensível e dinâmico**: o acréscimo de novas funcionalidades, ou mesmo a alteração de certos procedimentos não deveria alterar outros objetos, nem tão pouco exigir alteração dos mesmos.

- **Prover uma arquitetura com serviço de nomes**: além de oferecer uma forma única de acessar um objeto (através de seu nome), deve-se oferecer a possibilidade de criação de contextos para os nomes.
- **Pesquisas baseadas em nomes, atributos e relacionamentos**: além da possibilidade de buscar um objeto através de seu nome, é importante que exista outras formas de busca quando não há o conhecimento do nome do objeto. As formas alternativas podem estar baseadas nos seus atributos ou nos serviços que eles provêm.
- **Controle de acesso**: é importante poder controlar o que os objetos clientes podem conhecer e solicitar de um servidor.
- **Controle de concorrência**: visto que um objeto pode receber várias requisições simultaneamente ele deve manter a sua integridade e ser capaz de recuperar seu estado em caso de erro. O controle do paralelismo deve visar o acesso conciso às suas informações bem como manter a disponibilidade e o desempenho.
- **Transações**: algumas atividades, mesmo que formadas por várias operações diferentes, precisam ser atômicas, pois dessa forma o controle de transações se faz necessário.
- **Operações robustas e alta disponibilidade**: o sistema deve ser capaz de recuperar-se em caso de falha, bem como informar os seus clientes caso necessário.
- **Suporte e controle de versões.**
- **Notificação de eventos**: objetos que disparam eventos devem ter capacidade para conhecer aqueles objetos que precisam receber certos eventos por eles gerados.
- **Prover relacionamentos de semântica entre os objetos**: através de referências entre objetos.
- **Prover uma interface de programação (API – *Application Program Interface*) no mínimo para a linguagem C.**
- **Funções de administração e gerenciamento**: incluindo funções como administração à distância, geração e restauração de cópias de segurança.
- **Internacionalização**: possibilitar a adequação à língua e características locais do usuário como moeda, formato de data, endereço entre outros.
- **Procurar interoperar com padrões da indústria que sejam pertinentes.**

Buscando atender os objetivos técnicos acima descritos a OMG buscou na OMA descrever uma generalidade, um modelo mais abstrato, ou seja, de mais alto nível, o qual o CORBA concretiza. Pode-se dizer que, portanto, a OMA serve como um "tipo" e o CORBA uma "instância" desse tipo.

Existem algumas diferenças na literatura em relação à nomenclatura dos elementos da OMA. Muitos autores os citam como [MOW98] [MOW97] [OMG2000b]: CORBA ORB, CORBA *Services*, CORBA *Domains* e CORBA *Facilities*. Seguindo esse tipo de nomenclatura a distinção entre tipo e instância, citada anteriormente, no que se refere à OMA e ao CORBA se perde. Uma melhor definição para esses elementos pode ser [POP98]:

- **ORB** (*Object Request Broker*): é o responsável por prover o meio de comunicação entre os objetos, fornecendo transparência sobre o protocolo de transporte, plataforma, sistema operacional e localização.

- ***Services***: são objetos (aplicações) que fornecem serviços essenciais para as demais aplicações CORBA. Alguns exemplos são: serviço de nomes, de eventos, de transação, de segurança, etc.
- ***Domains***: são denominados serviços verticais, ligados às áreas de mercado, como de contabilidade.
- ***Facilities***: são as interfaces ou serviços, denominados de horizontais, que podem ser utilizados por vários *Domains*.
- **Aplicações**: representa a grande variedade de aplicações desenvolvidas como, por exemplo, *interfaces* comerciais proprietárias.

A FIGURA 3.1 traz uma representação gráfica da OMA.

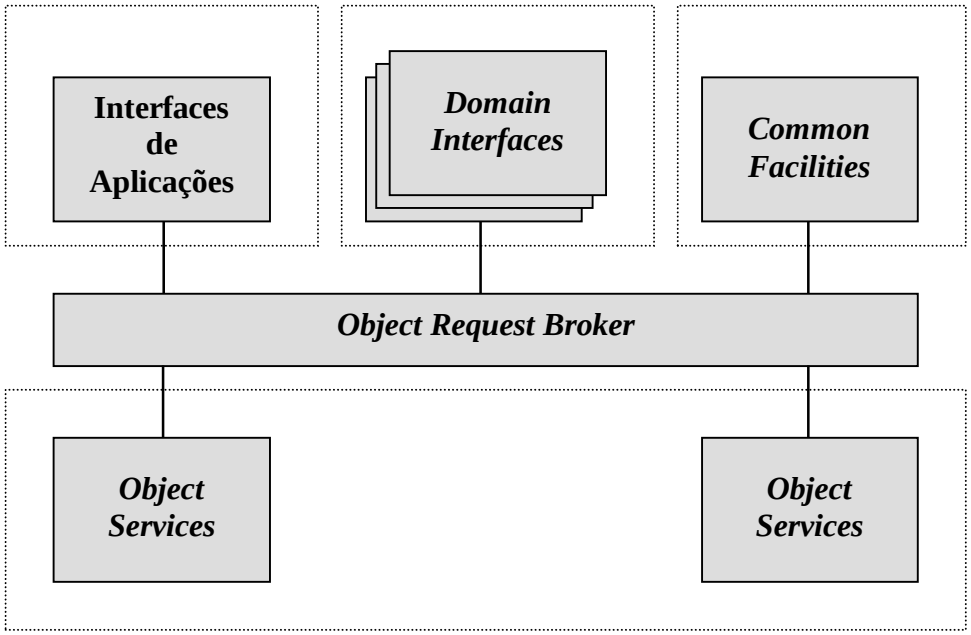


FIGURA 3.1 - Modelo estendido da OMA.

3.3 ORB – Object Request Broker

O ORB (*Object Request Broker*) é o mecanismo de comunicação padronizado pela arquitetura CORBA [MOW98]. Tem como objetivo fornecer uma forma única de acesso aos serviços, utilizando para isso um mecanismo orientado a objetos de chamada remota de procedimentos.

O ORB é, portanto, o *middleware* de comunicação que permite que os clientes façam requisições e recebam respostas de objetos servidores, os quais podem estar localizados localmente ou remotamente em relação ao cliente [ORF98].

Segundo as definições CORBA, foi introduzida uma linguagem de definição de interfaces (IDL – *Interface Definiction Language*), mapeamentos para diversas linguagens do mercado e APIs (*Application Program Interface*) para acesso ao ORB, deixando a sua forma de implementação livre. Isso, além de garantir uma liberdade para os desenvolvedores, assegura que um programa especificado para um ORB poderá ser parcialmente portável para qualquer outro ORB que seja compatível com as definições CORBA.

Vários serviços são oferecidos pelo ORB, tais como invocação de métodos de forma estática e dinâmica, mapeamentos da IDL para várias linguagens e formas de ativação de objetos. Na FIGURA 3.2 podem ser vistos alguns dos elementos, os quais formam a estrutura básica de um ORB compatível com as definições CORBA, responsáveis pelos serviços disponibilizados.

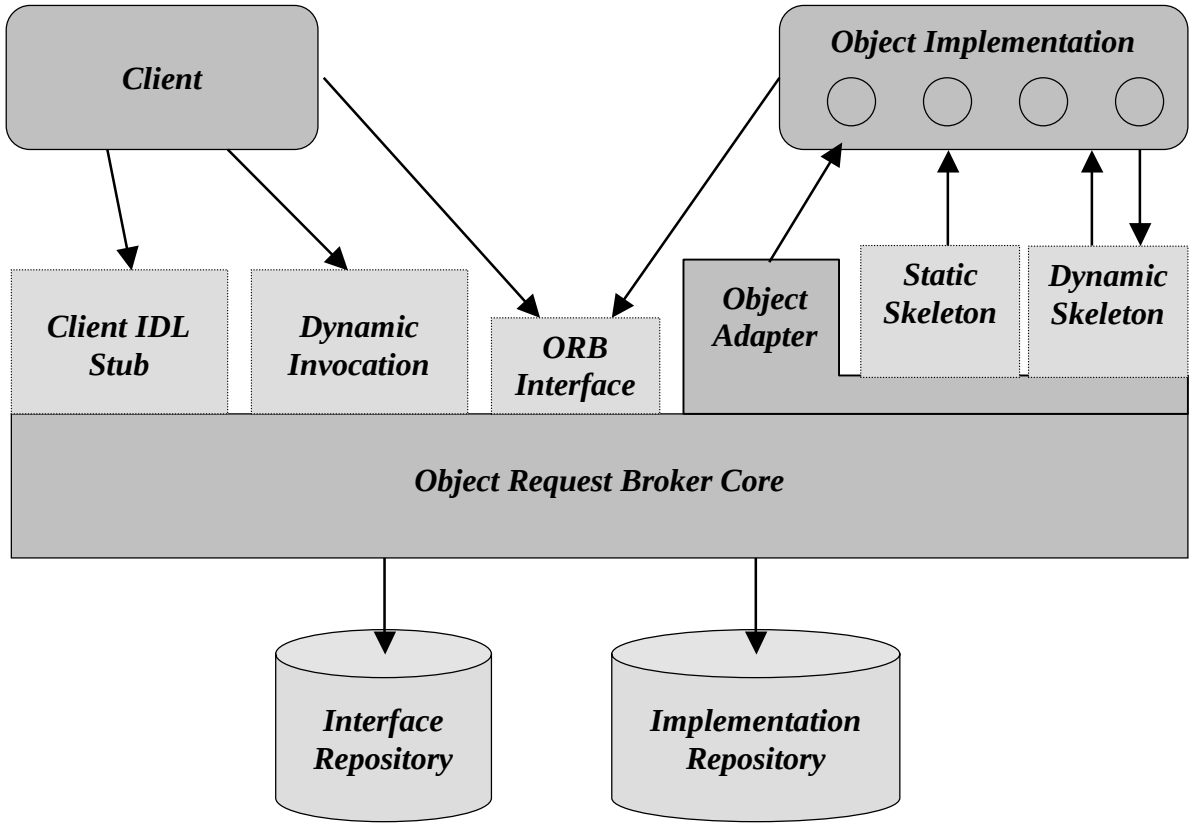


FIGURA 3.2 - Estrutura de um ORB.

Os elementos mostrados na FIGURA 3.2 serão detalhados nas seções seguintes, começando pela IDL, a qual define a forma como serão gerados os esqueletos básicos dos *stubs* dos clientes e dos *skeletons* dos servidores.

3.3.1 IDL – Interface Definition Language

A primeira versão padronizada da IDL foi lançada pela OMG em 1991. Poucas mudanças foram feitas nessa definição de modo que a IDL, com o avanço das versões do CORBA, tem se mantido praticamente inalterada. Visto que a IDL é a base da especificação de cada serviço definido pela OMG, esta estabilidade mais do que desejável, é necessária [MOW98]. Além disso, a IDL se tornou um padrão reconhecido pela ISO (*International Standards Organization*), o ISO DIS 14750, o que vem a reconhecer mais ainda os benefícios da IDL.

Como seu nome mesmo diz, a IDL é uma linguagem para definição de interfaces, ou seja, é a linguagem que define os contratos entre os objetos CORBA [POP98]. Não é, portanto, uma linguagem completa, não possui elementos de iteração, nem controle de fluxo, não sendo possível a implementação das interfaces definidas (nem esse é seu

objetivo). Possui um conjunto reduzido de tipos de dados, podendo construir tipos mais complexos a partir desses, acrescido da possibilidade de definição de escopo de nomes.

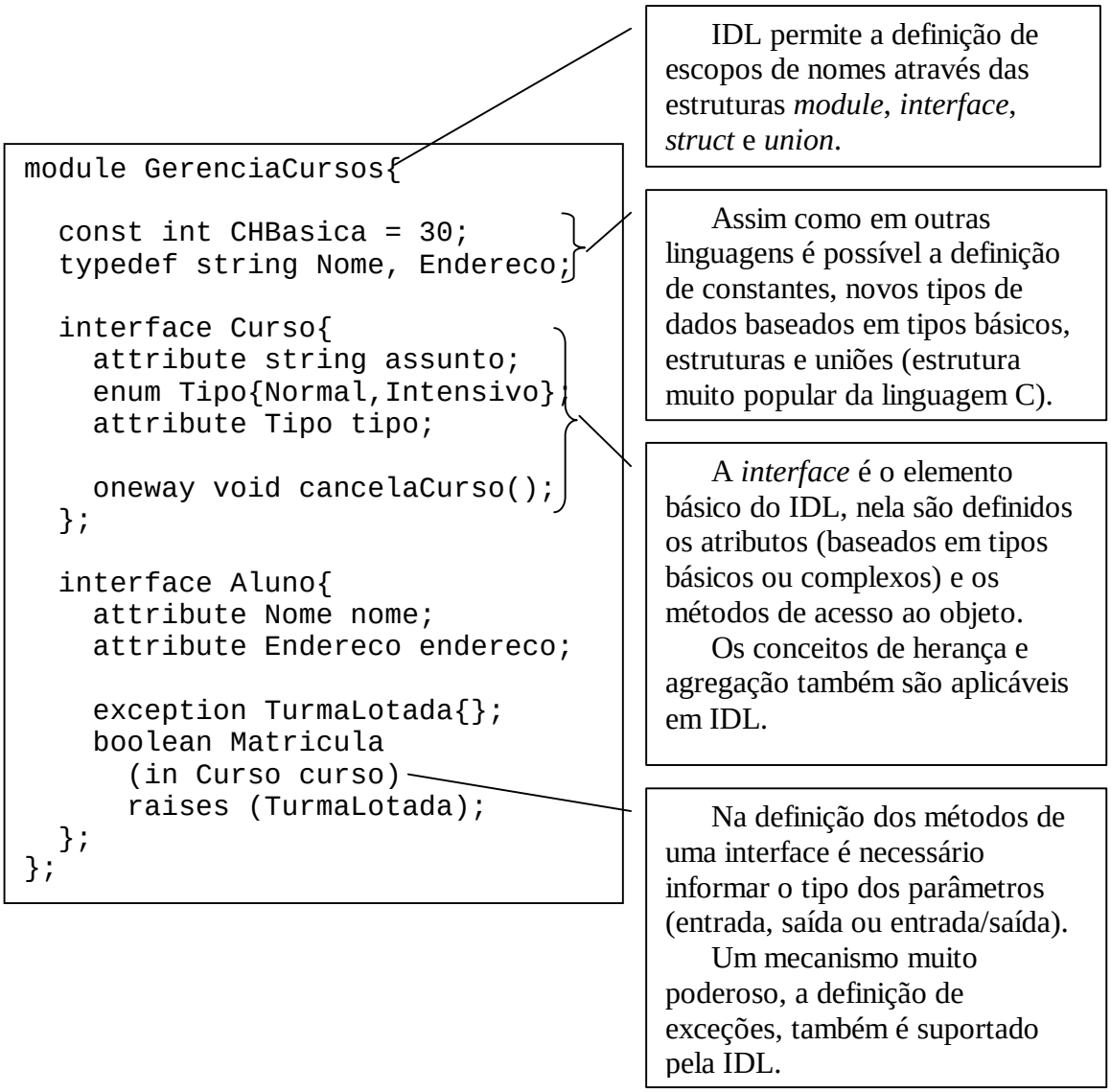


FIGURA 3.3 - Estruturas comumente encontradas em IDL.

Um compilador IDL normalmente é dividido em duas partes [POP98]: *front end* (FE) e *back end* (BE). O FE é o compilador IDL propriamente dito, é responsável pela checagem da correção na sintaxe do código IDL. Através de informações geradas pelo FE, o BE realiza o que se chama de mapeamento da IDL para outras linguagens, tais como C, C++, Pascal, Java e COBOL, entre outras. Portanto, o BE deve ser específico para cada linguagem, pois conhecendo as estruturas de cada linguagem ele é capaz de fazer a geração dos *stubs* do lado do cliente e dos *skeletons* do lado do servidor para as interfaces definidas no código IDL.

A FIGURA 3.3 apresenta algumas das estruturas básicas contidas na linguagem IDL. Muitas das estruturas e tipos de dados básicos definidos pela IDL também são encontrados no grande conjunto de linguagens de programação encontradas no mercado. Infelizmente, algumas das estruturas, como *sequence*, chamadas de função assíncrona (*oneway*), entre outros; não estão disponíveis. O grande desafio da OMG ao

definir o mapeamento entre IDL e as linguagens de mercado é encontrar a melhor forma de representar essas estruturas na linguagem alvo.

Além disso, outro ponto importante são os tipos de dados suportados em cada linguagem. Tomando-se como exemplo a linguagem Java [BRO00], a qual não suporta tipos sem sinal, como *unsigned int* existente em IDL, perdas de precisão podem ser encontradas no mapeamento e, tanto o compilador, como o ORB e o próprio programador, infelizmente, devem estar preparados para isso.

3.3.2 Invocação estática de métodos

Assim que uma definição de interface é compilada por um compilador IDL, são gerados dois elementos fundamentais: o *stub* do lado do cliente e o *skeleton* de implementação no lado do servidor. Esses elementos fornecem a interface estática para acesso aos serviços [POP98].

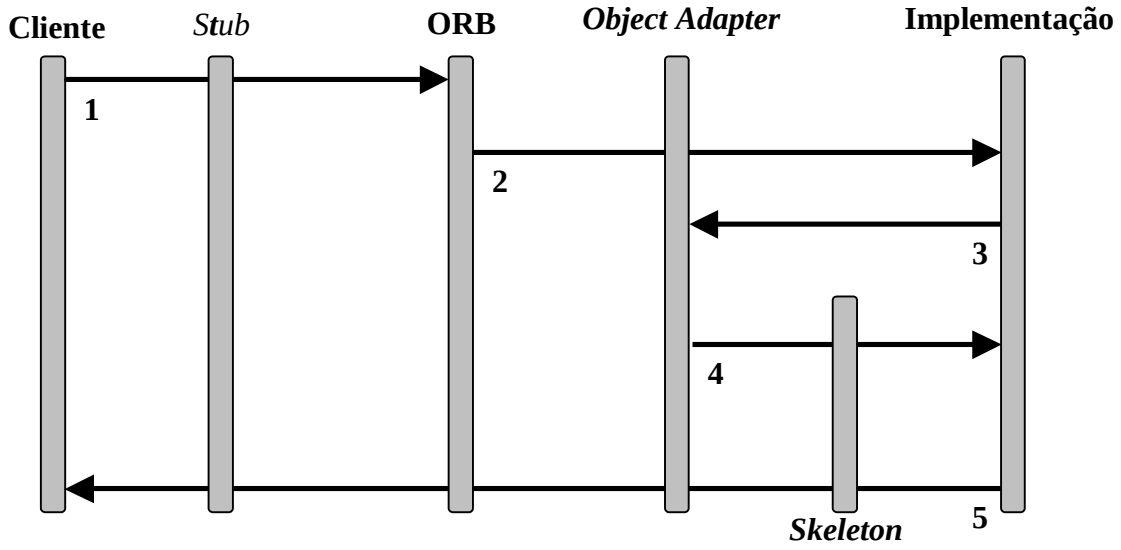


FIGURA 3.4 - Diagrama de interação típico.

O *stub* é um objeto local através do qual o cliente faz requisições. Pela perspectiva do cliente, o *stub* atua como uma chamada de método local, servindo como um intermediário para um objeto remoto. O compilador IDL gera todo o código necessário para esse *stub*, de forma que o programador não tenha que interferir. O código gerado é, portanto, uma implementação completa de um objeto (na linguagem para a qual o compilador IDL possui mapeamento), o qual inclui os procedimentos necessários para a codificação/decodificação dos parâmetros dos métodos, decodificação dos valores de retorno e as atividades relativas à localização do código e local correto do *skeleton* do servidor.

O *skeleton* é a implementação equivalente ao *stub* do lado do servidor. É gerado pelo compilador IDL o código necessário para a decodificação dos parâmetros dos métodos e codificação dos valores de retorno para a linguagem suportada. Entretanto, apenas um corpo de cada método é fornecido, o código complementar deve ser fornecido pelo programador.

Outro aspecto que diferencia o *skeleton* do *stub* é que o primeiro deve interagir com o *object adapter*, o qual, após efetuar a ativação do objeto servidor deve acessar

funções do *skeleton* para invocar os métodos da implementação do servidor. Isto pode ser visto no diagrama da FIGURA 3.4, que mostra a interação entre os elementos responsáveis por uma invocação de método estática.

3.3.3 Invocação dinâmica de métodos

Na abordagem apresentada anteriormente, para que um objeto (cliente) faça uma requisição para uma implementação (servidor) é necessário o prévio conhecimento da interface, ou seja, das assinaturas de cada um dos métodos da implementação.

Para muitas aplicações esta abordagem é suficiente. Entretanto, no momento em que se necessita de uma maior flexibilidade outro mecanismo se faz necessário. Assim, a invocação dinâmica de métodos do CORBA tem como objetivo permitir que, em tempo de execução, objetos sejam descobertos bem como seus métodos e seus parâmetros. Assim, através da DII (*Dynamic Invocation Interface*) um cliente pode alcançar qualquer objeto em tempo de execução e invocar seus métodos sem a necessidade de *stubs* pré-compilados [ORF98].

Os clientes podem descobrir implementações através de várias formas. Uma delas é realizar uma consulta pelo nome do objeto ao *Naming Service*, o qual pode fornecer uma referência ao objeto. Outra forma é procurar por objetos no *Trader Service*. Após a fase de descoberta do objeto é necessária a criação de um objeto intermediário, o pseudo-objeto CORBA *Request*, o qual fará a invocação do método no objeto remoto.

A invocação de métodos pode ser feita de três formas diferentes: síncrona (através do método *invoke*), assíncrona (através dos métodos *send_deferred* e *get_response*) e chamada de método sem retorno (método *send_oneway*).

Assumindo que uma referência para um objeto já foi obtida através de alguma das formas anteriormente citadas, pode-se resumir os passos para a invocação de um método desse objeto através da DII conforme a TABELA 3.2.

TABELA 3.2 - Processo para invocação dinâmica de métodos.

Ação	Método	Descrição
Obter o nome da interface	*get_interface*	Mesmo conhecendo-se a referência para o objeto é necessário conhecer sua interface. Cada objeto CORBA pode oferecer uma descrição de sua interface através de um objeto contido no *Interface Repository*, o objeto *InterfaceDef*.
Obter a descrição do método	*lookup_name describe*	O objeto InterfaceDef obtido no passo anterior serve como um ponto de acesso ao *Interface Repository*. Desta forma podem ser obtidas informações detalhadas sobre cada objeto contido no *Interface Repository* bem como sobre os métodos de cada um.
Criar a lista de argumentos	*create_list add_item*	Obtidos os detalhes sobre o método que se deseja invocar, é necessário construir uma lista com os nomes e valores dos argumentos necessários para o método em questão. CORBA define um tipo de objeto próprio para armazenar essa lista, o *NVList* (*Named Value List*).
Criar o *Request*	*create_request*	Já obtida uma referência para o objeto

		desejado e construída a lista de valores de argumentos para o método que será invocado pode-se, finalmente, criar um objeto *Request*. Através desse objeto é possível escolher uma das três formas de invocação do método remoto.
Invocar o método remoto	*invoke* *send_deferred* *get_response* *send_oneway*	O objeto *Request* oferece três formas de invocação dos métodos remotos: através do método *invoke* é feita uma requisição síncrona ao objeto remoto, isto é, o cliente ficará bloqueado até receber a resposta; uma requisição assíncrona pode ser feita através do método *send_deferred*, desta forma o cliente não ficará bloqueado e poderá obter a resposta para a requisição através do método *get_response*; o procedimento *send_oneway* permite a invocação de métodos remotos que não possuem retorno, dessa forma o cliente não precisa ficar bloqueado.

A invocação estática de métodos ainda é a mais natural, mais fácil de ser utilizada, que necessita da menor quantidade de código e que é implementada por todos os ORBs disponíveis. Por outro lado, a invocação dinâmica além de permitir que objetos sejam descobertos em tempo de execução, também permite a invocação de métodos conhecidos, assim como na invocação estática [MOW98]. O preço a se pagar por esta maior flexibilidade é um código mais complexo no lado do cliente e um retardo maior na chamada dos métodos.

3.3.4 Object Adapter

Pela filosofia do CORBA os clientes devem ser sempre o mais simples possível. Partindo desse princípio, para a visão dos clientes, os objetos servidores estão sempre executando e aptos a receber requisições [ORF98]. Entretanto, certos servidores podem conter muitos objetos, milhares até e seria impraticável manter sempre, todos os objetos em memória prontos a receber qualquer requisição, mesmo que elas sejam feitas apenas uma vez por dia, por exemplo.

Desta forma, é obrigação do ORB, no lado do servidor, manter a ilusão dos clientes de que todos os objetos estão ativos e, para isso, ele deve ser capaz de ativar ou desativar objetos quando necessário.

A forma alcançada para este controle do ciclo de vida dos objetos se dá através do *object adapter*, um pseudo-objeto CORBA, criado automaticamente pelo ORB, e que possui interfaces com o ORB e também com os *skeletons* do lado do servidor. Assim, o *object adapter* pode, além de ativar e desativar objetos, informar o estado do objeto servidor (se está ativo, se está apto a receber requisições, etc.).

A partir do CORBA 2.0 uma definição de *object adapter*, o BOA (*Basic Object Adapter*), pôde ser utilizado pelas implementações de ORBs existentes. A presença do BOA nos ORBs se tornou uma exigência, assim como: a existência de um repositório de implementações (com informações que descrevem os objetos); mecanismos para gerar e interpretar referências de objetos, ativar e desativar implementações e invocar métodos; invocação de métodos através de *skeletons*.

Ainda pela definição do CORBA 2.0, o BOA suporta quatro tipos de modos de ativação de objetos [ORF98]:

- *Shared server*: vários objetos podem estar contidos no mesmo processo (programa). O BOA ativa o servidor quando é recebida a primeira requisição para algum dos objetos implementados por aquele servidor. Quando pronto o servidor informa o BOA pela chamada do método *impl_is_ready*. A partir desse momento o BOA repassará todas as requisições para o servidor e não ativará outro servidor para aquela implementação. Quando estiver pronto para terminar o servidor informa o BOA pela chamada do método *dactivate_impl*, e este desativa todos os objetos que estavam rodando naquele processo.
- *Unshared server*: cada objeto está em um processo diferente, o servidor é ativado na primeira vez que uma requisição é feita para o objeto. O objeto informa que está ativo através do método *obj_is_ready*. *U*m novo servidor é iniciado quando é feita uma requisição para um objeto que ainda não está ativo, mesmo que um servidor para outro objeto com a mesma implementação esteja ativo.
- *Server per method*: um novo servidor é iniciado sempre a cada vez que uma nova requisição é feita. O servidor permanece ativo durante a execução do método. Muitos processos para o mesmo objeto (ou para um mesmo método de um mesmo objeto) podem estar ativos concorrentemente. É função do BOA ativar e desativar o servidor a cada requisição, desta forma o servidor não precisa informar quando está pronto ou desativado.
- *Persistent server*: o servidor é ativado fora do BOA e este é informado (pela chamada do método *impl_is_ready*) assim que o servidor estiver pronto para receber requisições. Assim que estiver pronto, o BOA trata todos as requisições subseqüentes como chamada a um *shared server*. Caso o servidor não esteja pronto um erro é retornado para a requisição.

Mesmo com as definições lançadas com o CORBA 2.0, o BOA ainda não ficou totalmente definido, e cada implementação de ORB do mercado introduziu suas próprias características para o BOA. Considerando a grande dificuldade que seria reconciliar as diversas versões de BOA existentes a OMG decidiu por lançar uma nova definição, melhorada e completa, de *object adapter*.

Assim o POA (*Portable Object Adapter*) é um substituto ao BOA, mas mantém muitas de suas características, corrigindo grande parte de seus problemas. Através das políticas de ativação adicionadas pelo POA (POA *Policies*), o controle sobre o comportamento dos servidores em tempo de execução aumentou em aspectos como: gerência do modelo de *threads*, unicidade de referência de um objeto, técnica de atribuição de identificadores, retenção e modelo de ativação.

Ainda, é importante apresentar os principais elementos com os quais o POA trata e a relação entre eles [HEN99]:

- *Servants*: uma aplicação pode criar *servants* para “encarnar” um ou mais objetos CORBA. São os *servants* que vão determinar o comportamento do objeto e que vão responder as requisições dos clientes.
- Referências a objetos (*object references*): o POA é responsável pela criação de referências para objetos CORBA. Referências podem ser criadas mesmo que nenhum *servant* tenha “encarnado” (*incarnate*) o objeto.

- Identificador de objeto (*object identifier*): cada objeto em um determinado POA possui um conjunto de bytes (*octet*) que o identifica unicamente, este nome pode ser atribuído pelo próprio POA ou pela aplicação.

Naturalmente, um objeto não existe até ter sido criado. Uma vez criado um objeto CORBA, é criada também uma referência para o mesmo, e a partir deste momento ele pode ser ativado ou desativado e receber requisições. Mas, para ter essas requisições processadas é necessário que um *servant* encarne (ou seja, dê corpo, dê comportamento a um objeto CORBA) esse objeto. Em um dado momento, apenas um *servant* pode estar encarnando um objeto CORBA. Entretanto, ao longo do tempo, vários *servants* podem encarnar um mesmo objeto CORBA, e finalmente, um *servant* pode ser *etherealized* (utilizando o termo da literatura) de modo a quebrar o vínculo entre ele e o objeto CORBA. É muito importante notar essa diferença entre os ciclos de vida dos objetos CORBA e seus servants (vistos na FIGURA 3.5), observando que, na literatura, os termos *create* e *destroy* são utilizados para objetos CORBA e os termos *incarnate* e *etherealize* são utilizados para *servants* [HEN99].

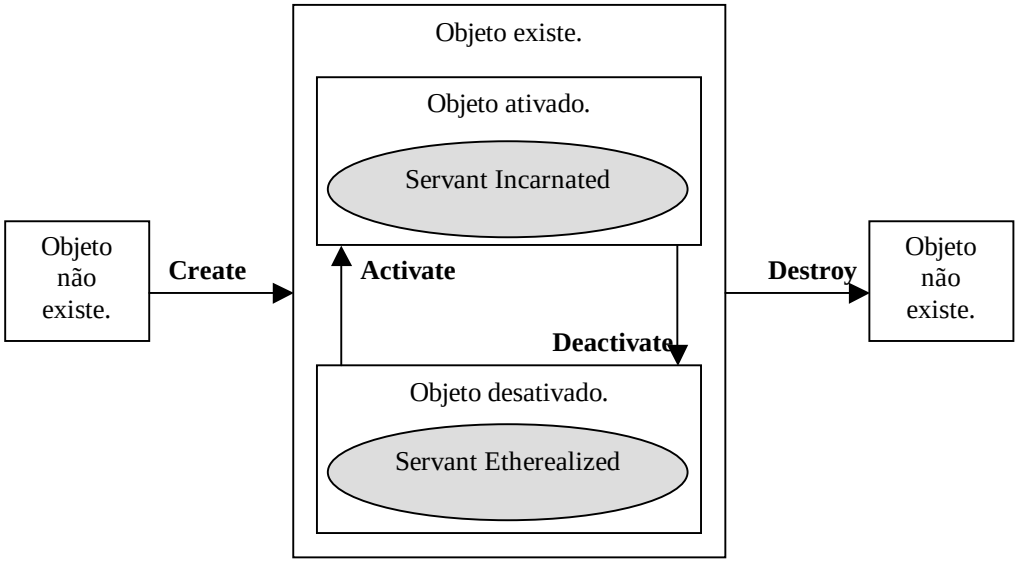


FIGURA 3.5 - Ciclos de vida de objetos CORBA e servants.

3.4 CORBA *Services*

CORBA *Services* são coleções de objetos (serviços) que acrescentam e complementam o ORB com funcionalidades básicas muito úteis para diversas aplicações. As *interfaces* desses serviços são especificadas em IDL, seguindo os padrões do CORBA [ORF98].

Esses serviços são como qualquer outro objeto CORBA, sem nenhuma relação especial com o ORB (com exceção de alguns poucos serviços como o de segurança), mas apresentam funcionalidades básicas, fundamentais e aplicáveis de várias formas e em várias aplicações.

Uma das vantagens advindas dos CORBA *Services* serem como qualquer outro objeto é a de que podem ser, portanto, implementados por qualquer um. Usuários de um ORB podem obter e utilizar um CORBA *Service* de outro ORB ou também podem fazer

suas próprias implementações no caso da ausência de um serviço. Ainda, a especificação dos serviços é tão simples quanto possível para permitir uma rápida implementação de um serviço ainda não presente em um ORB.

Segundo a OMA, todos os CORBA *Services* se encontram no mesmo nível de arquitetura, não apresentando uma organização completa dos serviços. Entretanto, os serviços definidos foram agrupados logicamente em categorias segundo suas características funcionais e arquiteturais, no que se denominou de arquitetura ORBOS (*Object Request Broker – Object Services*) [MOW98].

A TABELA 3.3 apresenta uma breve descrição dos serviços definidos pela OMG, organizados segundo a arquitetura ORBOS.

TABELA 3.3 - CORBA *Services* organizados segundo a ORBOS.

Categoria	Serviço	Descrição
Serviços de gerência de informações: serviços que se relacionam com a recuperação e manipulação de dados.	*Properties*	Permite que se associe valores nomeados (propriedades) com qualquer componente ou estado de um componente.
	Relationship	Oferece uma forma de criar associações dinâmicas entre componentes que não se conhecem. Pode ser utilizada para forçar integridade referencial, relação de agregação e outras formas de ligação entre componentes.
	Query	Fornece operações de consulta a objetos. É um super conjunto de SQL, baseado no SQL3 e no *Object Database Management Group's* (ODMG) *Object Query Language* (OQL).
	Externalization	Oferece meios padronizados de tratar a entrada e saída de dados de um componente através de mecanismos de *stream*.
	Persistence	Provê uma interface única para armazenar componentes em diferentes meios de armazenamento, como simples arquivos ou bancos de dados relacionais ou orientados a objetos.
	Collection	Fornece interfaces CORBA para criar e manipular as formas mais comuns de coleções.
Serviços de gerência de tarefas: objetos para gerenciar eventos e transações.	*Event*	Permite aos componentes dinamicamente registrarem ou "desregistrarem" seu interesse em eventos específicos.

Serviços de gerência do sistema: serviços para gerência de meta-dados, licença e ciclo de vida dos objetos.	*Concurrency*	Oferece meios para bloquear (obter *locks*) tanto em *threads* quanto em transações.
	Transaction	Oferece mecanismos de transação (inclusive aninhadas) e recuperação.
	Naming	Oferece meios para um componente encontrar outro pelo nome. Permite a criação de contextos de nomes.
	Life Cycle	Define operações para criar, copiar, mover e remover componentes.
	Licensing	Oferece operações para controle de acesso aos componentes. Permite controle de licenças por sessão, por nodo, por instância ou por *site*.
	Trader	Fornece um meio para os objetos publicarem seus serviços e procurarem por serviços registrados.
Serviços e elementos de infra-estrutura: reúne os serviços que são mais fortemente acoplados ao ORB.	*Security*	Ferramenta completa para segurança em objetos distribuídos. Suporta autenticação, lista de controle de acesso, confidencialidade, etc.
	Time	Oferece interfaces para sincronização de tempo em ambiente distribuído. Provê também operações para definir e gerenciar eventos controlados (ou disparados) por tempo.

Nas seções subseqüentes serão detalhados alguns dos serviços considerados mais importantes dentro do escopo do trabalho.

3.4.1 Naming Service

A forma mais fácil de identificação de um objeto é através de nomes. A função do serviço de nomes é, portanto, identificar objetos pelo seu nome quando é apresentada uma referência ao objeto, bem como o inverso, dado um nome de um objeto, retornar a referência ao mesmo.

Assim, existem duas operações fundamentais no serviço de nomes: *bind* e *resolve*. A operação de *bind* é invocada pelo objeto servidor para registrar um nome para uma referência a objeto e também associar esta entrada em um contexto de nomes. Já a operação de *resolve* é invocada por um objeto cliente, o qual deseja obter uma referência a um objeto, informando apenas o seu nome.

Outra função do serviço de nomes é fornecer contextos de nomes. Um contexto de nomes é organizado como uma árvore, podendo ter uma ou mais raízes. Os nodos internos da árvore representam os contextos de nomes (os quais definem escopos e sub-

escopos do espaço de nomes), e por sua vez, as folhas representam os nomes das instâncias atuais dos objetos.

Dessa forma, em um dado contexto, um nome de objeto é único. Assim, quando um objeto tenta invocar um *bind* com um nome já existentes naquele dado contexto, uma exceção é gerada. Ainda, é gerada uma exceção quando um objeto tenta resolver um nome que não foi registrado. Uma forma de evitar isso é através de métodos de varredura (caminhamento) pelas estruturas de nomes registrados em um dado diretório do serviço de nomes.

O *Naming Service* está presente na maioria dos ORBs. Ele se tornou um importante serviço para o desenvolvimento de aplicações, visto que permite às aplicações delegar o gerenciamento (bem como o seu armazenamento) das referências a objetos, para um serviço separado, estando assim sempre disponíveis em um serviço comum e acessível a todos os clientes.

3.4.2 Event Service

O serviço de eventos define interfaces genéricas para passagem de eventos, informações, entre múltiplas fontes e múltiplos receptores [MOW98]. Uma das grandes vantagens no serviço de eventos do CORBA é a existência de um canal de eventos (*event channel*), o qual permite a geração e recepção de eventos sem que os objetos que enviam e os que recebem tenham que se comunicar diretamente, e mais, nem mesmo eles têm a necessidade de conhecer um ao outro.

De forma a estar apto a gerar eventos um objeto deve se registrar no canal como um gerador de eventos, informando quais são os mesmos. Da mesma forma, um objeto deve se registrar no canal para demonstrar seu interesse em tomar conhecimento de determinados eventos.

Podem existir várias alternativas de interação entre o canal e os geradores e consumidores de eventos. Os dois modelos principais de interação são:

- *Push*: neste caso o gerador de eventos informa (envia o evento) todos os consumidores que já estavam registrados a receber aquele evento, através do canal. O canal armazena temporariamente todos os eventos gerados a fim de garantir o recebimento dos mesmos por parte de todos os consumidores.
- *Pull*: nesta alternativa é o consumidor que toma a iniciativa de receber os eventos. Existe uma opção bloqueante, na qual o consumidor invoca o método pull do canal. Na ausência de evento, a *thread* do consumidor fica esperando até a geração de um. De forma a permitir uma alternativa não-bloqueante existe um método *try-pull*, que verifica a existência de um evento específico. Se o retorno for verdadeiro o consumidor pode invocar o método *pull* e obter a informação do evento, caso contrário poderá continuar o processamento e testar a chegada do evento em outro momento.

A FIGURA 3.6 mostra esses dois métodos de interação, os quais podem existir simultaneamente. Assim como um gerador de eventos também pode ser um consumidor de eventos.

Existem duas interfaces para esses métodos: uma onde os eventos não são tipados (eles assumem o tipo *any* do CORBA) e outra onde os eventos possuem tipos definidos pelo programador.

Na primeira, o gerador e o consumidor devem entrar em acordo (especificando convenções previamente) sobre o formato das informações dos eventos. Já na segunda, os tipos dos eventos são checados estaticamente, em tempo de compilação, o que é uma vantagem reforçada com o uso de linguagens fortemente tipadas.

O serviço de eventos pode ter várias aplicações além da originalmente definida, como comunicação *multicast* e ativações de gerações de eventos baseadas em tempo, necessárias em sistemas de tempo real.

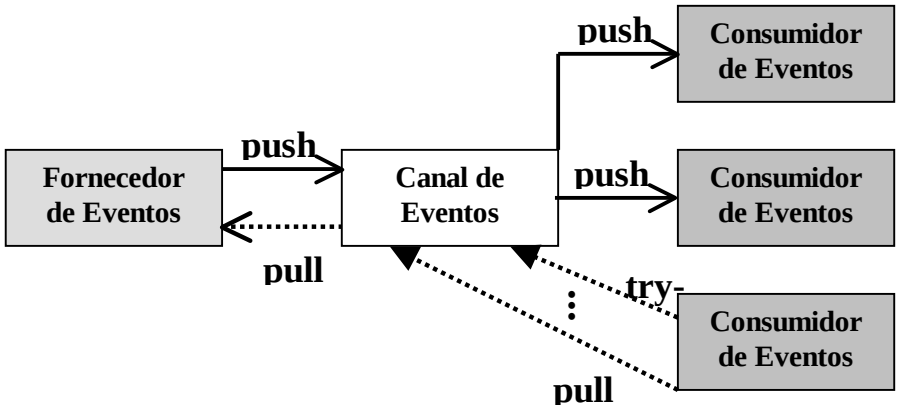


FIGURA 3.6 - Alternativas de interação para troca de eventos.

Na FIGURA 3.6 podem ser encontrados os participantes da arquitetura do serviço de eventos CORBA:

- ***suppliers*** (fornecedores) e ***consumers*** (consumidores): os consumidores são os alvos dos eventos gerados pelos fornecedores. Eles podem assumir uma postura ativa ou passiva conforme os modelos de interação citados anteriormente.
- ***event channel*** (canal de eventos): é o coração do serviço de eventos, funcionando como um intermediário entre os fornecedores e os consumidores, gerenciando referências para os mesmos. Ele se assemelha a um consumidor *"proxy"* para os verdadeiros fornecedores de um lado e como um fornecedor *"proxy"* para os verdadeiros consumidores do outro.

Os fornecedores usam o canal de eventos para enviar (*push*) dados aos consumidores e estes, por sua vez podem solicitar (*pull*) dados dos fornecedores também através do canal de eventos. O serviço de eventos acrescenta ao CORBA a possibilidade de uma comunicação assíncrona menos restritiva que a forma normal *two-way*. Ainda há a possibilidade de utilização do canal de eventos para a implementação de comunicação em grupo, este servindo como um replicador (*broadcaster* ou *multicaster*) que repassa mensagens de um ou mais fornecedores para múltiplos consumidores.

3.5 CORBA *Facilities* e CORBA *Domains*

Os CORBA *Services*, vistos na seção anterior, são as interfaces fundamentais que foram padronizadas. Essas interfaces podem ser utilizadas para a construção de

aplicações complexas, possuindo papel semelhante às chamadas de sistema de um sistema operacional. As aplicações, por sua vez, são as tecnologias especializadas que não são padronizadas pela OMG. Já os outros elementos contidos na arquitetura CORBA são incluídos nas CORBA *Facilities* e CORBA *Domains*.

Enquanto os CORBA *Services*, normalmente, não dependem um do outro, os CORBA *Facilities* representam outra camada de tecnologia que pode reutilizar, estender ou herdar características dos CORBA *Services* [MOW98]. CORBA *Facilities*, que também são chamadas de *Horizontal Facilities*, representam, portanto, capacidades de mais alto nível as quais buscam a interoperabilidade das aplicações oferecendo serviços mais complexos em áreas como gerência da interface com o usuário, gerência de informações, gerência do sistema e gerência de tarefas.

Por sua vez, CORBA *Domains* são as áreas verticais do mercado, por exemplo: serviços financeiros, de saúde, de manufatura, telecomunicações e objetos de negócios. Ao contrário dos CORBA *Facilities*, CORBA *Domains* não buscam a interoperabilidade entre os domínios, mas apenas dentro do seu próprio domínio, assumindo, dessa forma, uma visão mais vertical.

Assim, a última categoria de objetos, os quais não são padronizados pela OMG, os objetos de aplicação, podem utilizar as características fornecidas tanto pelos CORBA *Services* como pelos CORBA *Facilities* e CORBA *Domains*. Isto visa um desenvolvimento mais rápido e seguro através da utilização de objetos já desenvolvidos para um conjunto de atividades que se mostraram necessárias para as mais variadas aplicações. A FIGURA 3.7 apresenta graficamente a relação de dependência entre esses níveis da arquitetura CORBA.

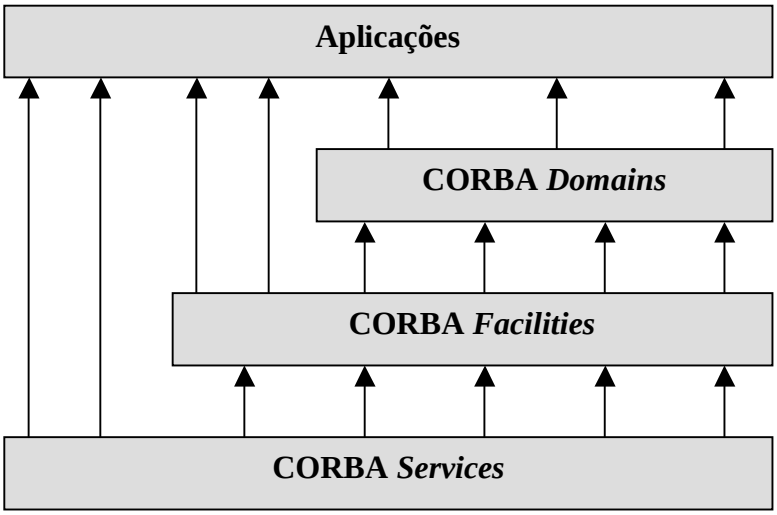


FIGURA 3.7 - Relação das especificações da OMG.

3.6 Considerações Finais

Definido pela OMG, que organiza um conjunto com mais de 800 empresas, CORBA é, atualmente, uma das arquiteturas para desenvolvimento de aplicações em sistemas distribuídos mais completas.

Ela busca a independência de plataforma, comunicação e linguagem. É baseada em um *middleware* de comunicação, um barramento (o ORB) que realiza a codificação dos parâmetros das funções e efetua a invocação dos métodos. Através da criação de

uma linguagem própria (a IDL) para a definição de interfaces de objetos CORBA consegue-se a integração até mesmo com aplicações que não foram originalmente projetadas para ambientes distribuídos.

Uma das grandes vantagens do CORBA está em sua flexibilidade. Esta característica decorre do modelo de arquitetura adotado para a definição de seus serviços. Os CORBA *Services* foram definidos através da IDL como objetos CORBA normais, e, desta forma, podem ser entendidos ou especializados conforme as necessidades das aplicações. Inicialmente foram definidos aproximadamente 15 serviços incluindo: serviço de nomes, de eventos, de tempo, de consulta, de segurança, de transação, entre outros.

Muitos desses já estão implementados (ou estão em fase de conclusão) na maioria das distribuições de ORBs do mercado, e outros estão sendo definidos pela OMG.

Em um nível organizacional mais complexo estão os CORBA *Facilities* e CORBA *Domains*, os quais consistem em conjuntos de serviços especializados em determinadas áreas, que são baseados em CORBA *Services* já existentes.

CORBA está, gradativamente, sendo adotado como padrão para o desenvolvimento de sistemas distribuídos, visto que o conjunto de serviços e a liberdade de extensão de suas definições abrem novas possibilidades. Uma das iniciativas já padronizadas e agora em fase de ampliação é o RT-CORBA (*Real Time* CORBA), o qual será visto com mais detalhes no Capítulo 3 deste trabalho.

4 RT-CORBA e o ORB TAO

A partir da década de 80 a tecnologia de sistemas distribuídos se tornou cada vez mais popular na área da computação de tempo real. Esta tendência se tornou ainda mais evidente e acelerada na década de 90, através de inúmeras pesquisas, desenvolvimento de novos padrões e congressos na área (como o *International Symposium on Object Oriented Real Time Distributed Computing* – ISORC – iniciado em 1998 e com periodicidade anual) [SHO2000].

Isso está muito relacionado ao fato de que a computação distribuída pode trazer [SCH2000]: aumento de desempenho através do multi-processamento; confiabilidade e disponibilidade maiores através da replicação; escalabilidade e portabilidade através da modularidade; e redução de custos através da reusabilidade dos objetos e dos sistemas abertos.

Visto que o CORBA se tornou o padrão de *middleware* para a computação com objetos distribuídos certamente a sua utilização traria muitas das vantagens acima citadas para o desenvolvimento de aplicações de tempo real. Entretanto, o CORBA 2.x não se enquadra em todas as exigências das aplicações para alto desempenho e tempo real, principalmente pelas seguintes razões [SCH2000] [OMG2000c]:

- **Ausência de interfaces para definir QoS (*Quality of Service*):** no CORBA 2.x não há interfaces para que os clientes e servidores possam definir as suas exigências de QoS como prioridades das requisições, período de execução de atividades cíclicas e políticas de admissão;
- **Ausência de garantias de QoS:** nos ORBs convencionais não há garantias com respeito às exigências de QoS fim-a-fim. Podem ocorrer inversões de prioridades de tarefas, consumo exagerado de recursos por uma atividade, etc;
- **Ausência de características de programação para tempo real:** ainda, no CORBA 2.x não são definidas algumas características essenciais para programação de tempo real como notificação quando a camada de transporte efetua controle de fluxo ou especificação de operações dependentes de tempo;
- **Ausência de otimizações para alto desempenho e previsibilidade:** ORBs convencionais trazem muita sobrecarga no desempenho e latência, bem como muitos pontos de inversão de prioridade e fontes de não-determinismo. Muitos desses problemas provêm de: camadas de apresentação não-otimizadas com muitas cópias de dados, *"bufferização"* interna com comportamento não-uniforme, algoritmos de demultiplexação e despacho ineficientes e principalmente, falta de integração com sistemas operacionais de tempo real e mecanismos de QoS nas redes.

Assim, em virtude dessa falta de suporte do CORBA para aplicações de tempo real, a OMG criou em 1995, um grupo de trabalho com o objetivo de estender as especificações CORBA. Em outubro de 1998 cinco propostas apresentadas (pelas empresas Alcatel, Highlander, Lockheed, Northern e OIS) foram unidas em um único documento [OMG2000c], o RT-CORBA. As seções subseqüentes desse capítulo abordam esse padrão, o RT-CORBA, que atualmente encontra-se em desenvolvimento sua versão 2, a qual passará a integrar o CORBA 3.

4.1 Visão Geral do padrão RT – CORBA

A especificação *Real Time* CORBA (RT-CORBA) define características padronizadas para suportar previsibilidade fim-a-fim na invocação de operações de aplicações CORBA com prioridade fixa (a OMG já está trabalhando em uma padronização para escalonamento dinâmico, baseados em deadline ou valor) [SCH2000a]. Esta especificação estende os padrões CORBA existentes e o serviço de mensagens recentemente adotado, de forma opcional, sendo possível, dessa forma, portar aplicações CORBA normais para um RT-ORB sem necessidade de modificações de código. Essas aplicações simplesmente não fariam uso das extensões de tempo real existentes [MON99].

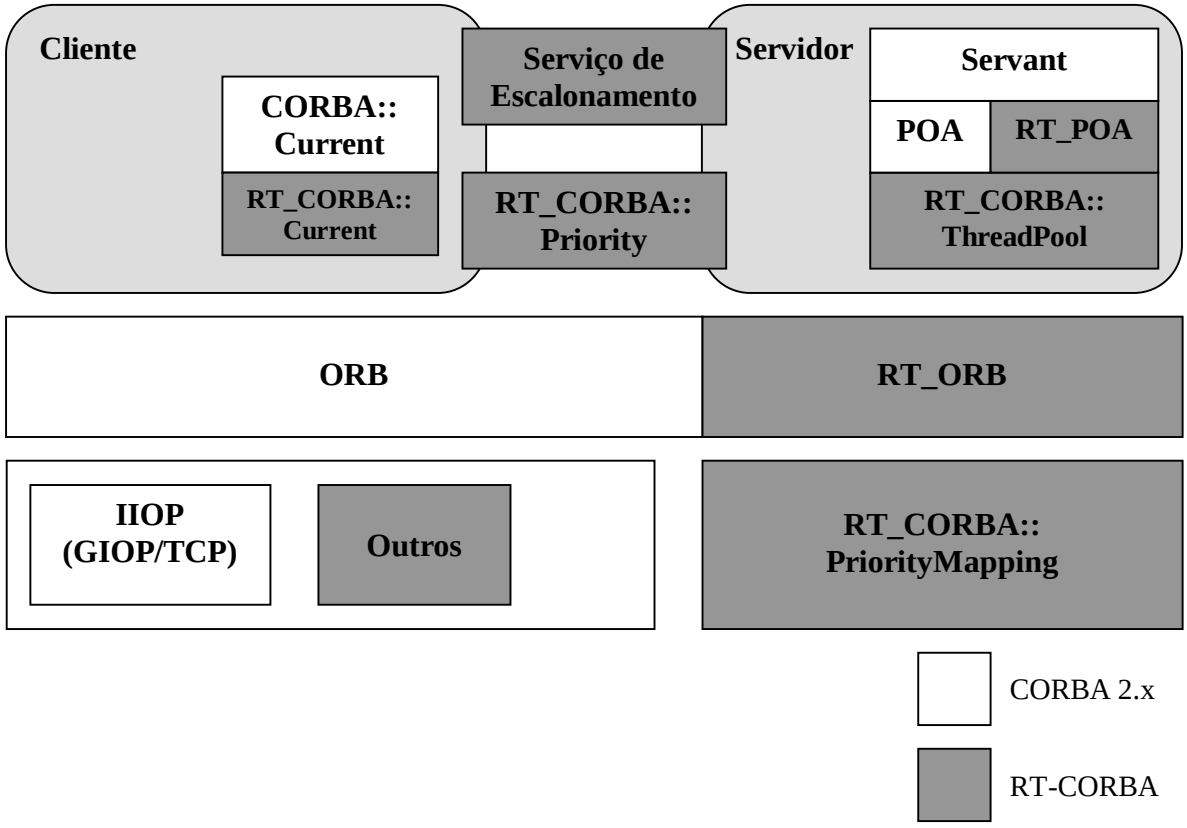


FIGURA 4.1 - Principais componentes do RT-CORBA.

O RT-CORBA identifica capacidades que podem ser verticalmente (da camada de rede para a camada de aplicação e vice-versa) e horizontalmente (fim-a-fim) integradas e gerenciadas por um ORB para garantir previsibilidade nas atividades entre clientes e servidores CORBA. Seguem abaixo algumas dessas capacidades [SCH2000a]:

- **Gerenciamento dos recursos de comunicação:** o RT-CORBA deve prover políticas e mecanismos para gerenciar a infra-estrutura de comunicação, desde a escolha de uma conexão até a exploração das características avançadas de QoS;
- **Mecanismos de escalonamento do sistema operacional:** ORBs exploram os mecanismos do sistema operacional para escalonar as atividades no nível da aplicação. O RT-CORBA se preocupa, inicialmente, com sistemas de tempo

real de prioridade fixa. Dessa forma, estes mecanismos correspondem a gerência de prioridades das *threads* escalonadas pelo sistema operacional;

- **ORB de tempo real:** um ORB de tempo real deve prover interfaces padronizadas para permitir às aplicações especificar suas exigências de recursos para o ORB, além de fornecer comunicação entre os clientes e servidores de forma transparente;
- **Serviços e aplicações de tempo real:** um ORB de tempo real também deve criar um ambiente eficiente, escalonável e previsível fim-a–fim para serviços e aplicações de alto nível.

Para gerenciar estas capacidades o RT-CORBA define interfaces padronizadas e políticas de QoS que permitem às aplicações controlar e configurar recursos (como de processador, memória e comunicação). Os principais componentes do RT-CORBA podem ser vistos na FIGURA 4.1 [OMG2000e].

Nas seções seguintes serão vistos esses principais componentes do RT-CORBA.

4.2 Controle de Recursos

As aplicações de tempo real precisam formas de controlar os recursos utilizados pelo ORB tanto do lado do cliente quanto do lado do servidor. O gerenciamento dos recursos pode estar classificado em duas categorias fundamentais [OMG2000c]:

- Alguns recursos são alocados estaticamente ou basicamente durante a inicialização do ORB (como *threads*, portas de comunicação, *buffers*). Não há sobrecarga em tempo de execução visto que uma vez alocados estes recursos não serão liberados;
- Outros recursos são alocados dinamicamente (como *buffers* de requisições, parâmetros de funções, etc.). A necessidade de o ORB alocar e desalocar recursos dinamicamente, em tempo de execução, acarreta dois problemas: imprevisibilidade e possibilidade de exaustão dos recursos.

A pré-alocação de recursos é um ponto importante para garantir a previsibilidade fim-a-fim, muito necessária em aplicações de tempo real. Para gerenciar e controlar os recursos, o *real time* CORBA define algumas interfaces de programação como: *Real Time Thread* API, *Request Queue* API, *Transport Management* API e *Buffer Management* API.

Alguns dos pontos principais no gerenciamento de recursos no RT-CORBA são discutidos nas próximas seções.

4.2.1 Threads

O *real time* CORBA utiliza as *threads* como sendo as entidades escalonáveis básicas. Geralmente, elas representam uma seqüência de comandos em um nodo. O RT-CORBA especifica interfaces através das quais as características das *threads* podem ser manipuladas.

As *threads* são utilizadas em dois níveis nas aplicações CORBA. Primeiro, elas são utilizadas pelo ORB para esperar requisições, despachar e executar o código do

servidor. Segundo, elas podem ser utilizadas pelas aplicações para seus próprios propósitos.

Uma aplicação de tempo real deve ser capaz de especificar as características e o número de *threads* que o ORB vai utilizar para despachar as requisições.

A *Real Time Thread* API está organizada em quarto partes:

- ***Thread Attributes***: definem as características das *threads* e especificam o que pode ser configurado. Os atributos são especificados como uma interface CORBA, a qual apresenta uma API genérica que pode ser estendida para implementações específicas de *threads*;
- ***Thread Management***: oferece uma interface para controlar a criação, destruição e mudança de atributos das *threads* em tempo de execução. As funções de suspender e reiniciar uma *thread* não foram propostas na API genérica porque são funcionalidades não-portáveis, cuja semântica seria difícil de especificar;
- ***Thread Specific Data***: é um mecanismo que permite às aplicações associar dados específicos a uma *thread*. Assim, normalmente, pode-se obter um aumento de desempenho e simplificação das aplicações *multi-thread* reduzindo a sobrecarga de trabalho existente nas operações de adquirir e liberar *locks*, isto porque os dados não precisam ficar protegidos, visto que são específicos de cada *thread*.
- As *threads* podem ser agrupadas em um ***Thread Pool*** o que permite às aplicações controlar as *threads* do ORB.

4.2.2 Estoque de Threads (Thread Pools)

Threads podem ser agrupadas para formar um estoque de *threads* (*Thread Pool*), o qual representa um conjunto de *threads* com aproximadamente as mesmas características (como por exemplo: classe de escalonamento, atributos, tamanho de pilha, entre outros). Um estoque de *threads* pode ser utilizado pelo ORB para que [OMG2000c]: a camada de transporte receba requisições; o *Object Adapter* despache as requisições; o cliente receba respostas e processe requisições assíncronas.

A especificação RT-CORBA define dois diferentes tipos de estoque de *threads* [SCH2000a]:

- **Estoque de *threads* sem faixas:** o modelo mais simples de estoque de *threads* definido pelo RT-CORBA permite ao desenvolvedor controlar o nível de concorrência no servidor e na aplicação. Um estoque de *threads* é criado com um número fixo de *threads* alocadas estaticamente, que consomem recursos mesmo que não estejam sendo utilizadas. Além disso, são fornecidos recursos para que o estoque possa crescer dinamicamente de modo a conseguir tratar possíveis rajadas de requisições dos clientes. As aplicações podem definir o número padrão de *threads* estáticas criadas inicialmente, o número máximo de *threads* que podem ser criadas dinamicamente e o valor padrão de sua prioridade. No momento da chegada de uma requisição no servidor automaticamente uma *thread* que se encontra disponível é destinada ao processamento da requisição. Caso, em um determinado momento, todas as *threads* que foram inicialmente alocadas se encontrem ocupadas, pode-se criar

uma adicional para atender a requisição até que o número máximo de *threads* determinado na criação do estoque seja alcançado. No momento em que se chegou ao número máximo de *threads* dinâmicas qualquer requisição adicional que chegue deverá ser armazenada em uma fila de requisições, ser descartada ou deverá se tomar outra ação determinada pela política de controle de fluxo do servidor.

- **Estoque de *threads* com faixas:** muitas aplicações de tempo real associam estaticamente prioridades CORBA a estoque de *threads*. Dessa forma, é desejável que se possa particionar as *threads* em subconjuntos (*pools*), cada um com uma prioridade diferente. Assim, nos estoques de *threads* com faixas especificados pelo RT-CORBA o servidor pode especificar, assim como no tipo anterior, a prioridade, o número de *threads* estáticas e o número máximo de *threads* dinâmicas. A diferença básica desse tipo reside na possibilidade de configurar os estoques de *threads* de forma que conjuntos de *threads* de alta prioridade possam solicitar recursos (*threads*) daqueles de mais baixa prioridade. Ou seja, neste caso, no momento em que uma requisição encontre o pool de *threads* com escassez de recursos (possuindo, portanto, todas as possíveis *threads* já criadas), este pode solicitar uma *thread*, a qual se encontra disponível, de um estoque (de prioridade inferior) para atender a requisição. Quando a *thread* é emprestada sua prioridade é alterada temporariamente, sendo retornada à sua configuração inicial quando o processamento é completado. Com a possibilidade de "empréstimo" de *threads* de um estoque para outro diminui-se o risco de inversão de prioridades nas requisições.

4.2.3 Prioridades

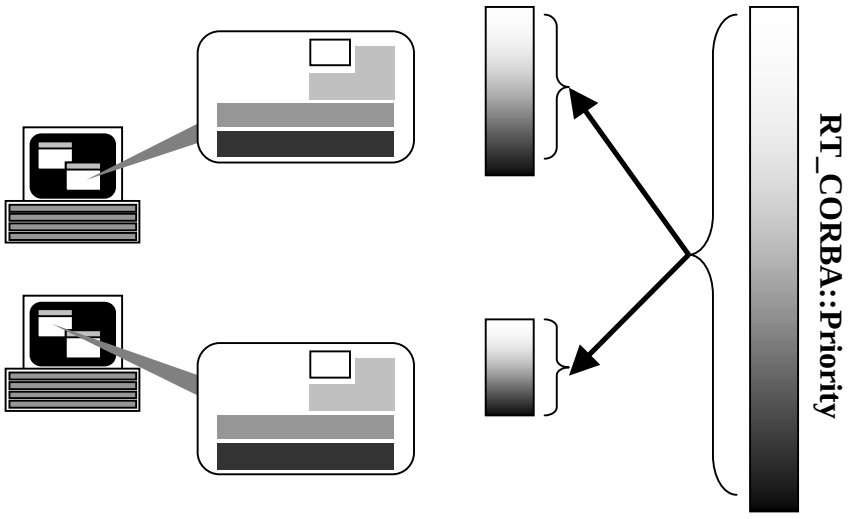


FIGURA 4.2 - Mapeamento de prioridades CORBA para prioridade nativas.

A especificação RT-CORBA define dois tipos de prioridades: CORBA e nativa. Cada operação, *one-way ou two-way*, pode ter associada uma prioridade CORBA que pode variar em uma escala de 0 a 32767 (valor inteiro de 16 bits). Cada ORB ao longo de um caminho de comunicação mapeia a prioridade CORBA em prioridade nativa (esta

pode ser diferente e única para cada sistema). Na FIGURA 4.2 pode ser visto um caso possível desse mapeamento de prioridades.

Além destes dois tipos de prioridades, o RT-CORBA especifica dois modelos para a utilização e definição de prioridades [SCH2000a] [OMG2000d]:

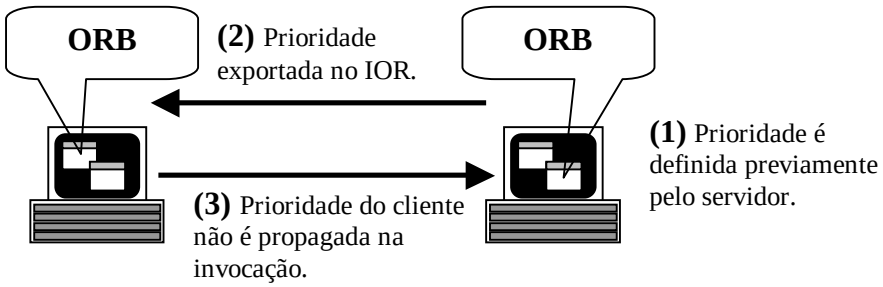


FIGURA 4.3 - Modelo de prioridade definida pelo servidor.

- **Prioridade definida pelo servidor:** permite ao servidor indicar a prioridade na qual as invocações serão feitas em um objeto em particular. Neste modelo a prioridade é definida a priori conforme o valor definido no POA pelo qual o objeto foi ativado. Uma única prioridade é codificada na referência do objeto, a qual é então publicada para o cliente. Este modelo pode ser visto na FIGURA 4.3. Visto que o cliente pode conhecer a prioridade definida pelo servidor para cada objeto ele pode ser capaz de implementar as conexões e invocações de forma a respeitar a prioridade definida pelo servidor.

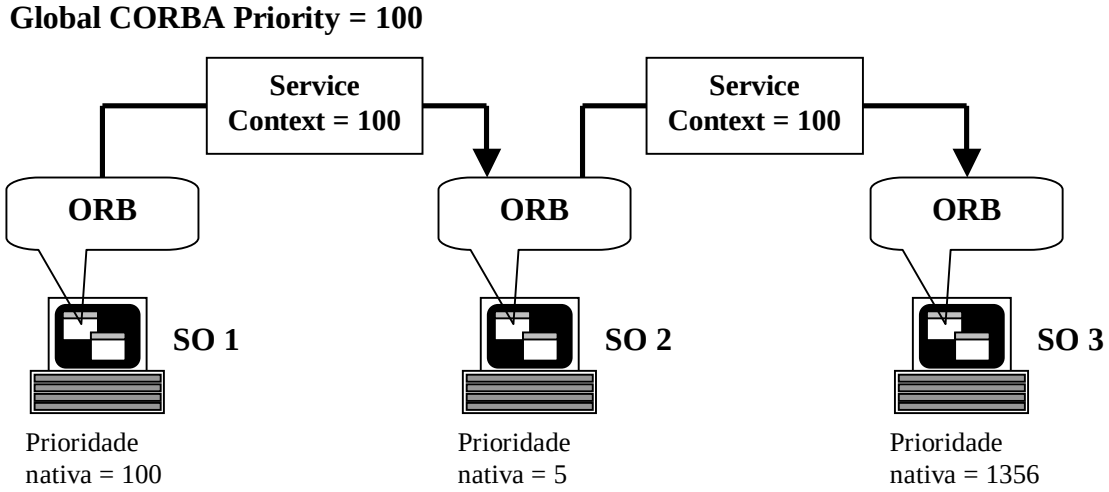


FIGURA 4.4 - Modelo de prioridade definida pelo cliente.

- **Prioridade definida pelo cliente:** uma forma de prevenir inversão de prioridades no servidor é este seguir as prioridades definidas pelo cliente. Neste modelo cada invocação carrega a prioridade CORBA do cliente. Ao longo de cada ORB entre o cliente e o servidor, este mapeia a prioridade CORBA para uma prioridade nativa e processa a solicitação (caso a operação for *two-way*, a resposta também seguirá a mesma prioridade). A FIGURA 4.4 mostra um caso de requisição, no qual a prioridade é propagada pelo cliente e

cada servidor ao longo do caminho mapeia a prioridade CORBA para a prioridade nativa de cada sistema operacional [SCH2000b].

4.2.4 Fila de requisições e controle de fluxo

Normalmente, as aplicações de tempo real são sensíveis aos tempos de resposta e latência existentes nas filas de recepção de requisições. Entretanto, a existência de filas na recepção é inevitável devido à impossibilidade de alocar uma nova *thread* para cada nova requisição recebida em determinados momentos ao longo da execução do sistema [OMG2000d].

Assim, para o correto processamento das requisições pelo ORB é necessária a existência de filas de recepção basicamente porque [OMG2000c]:

- De uma forma simplificada, uma fila é necessária para organizar as requisições. Entre as várias formas possíveis de organizar as requisições, pode-se tomar como exemplo, a organização por suas prioridades ou também de acordo com outras definições de QoS;
- Como as *threads* são recursos escassos, em momentos de carga alta, muitas vezes é mais vantajoso enfileirar as requisições do que criar novas *threads*;
- Uma fila de requisições é o primeiro mecanismo para controle de fluxo.

As definições do RT-CORBA para a *Request Queue* API são genéricas e não definem uma política em particular. São definidas apenas operações básicas para inserir e retirar requisições da fila e obter o número de requisições pendentes. Assim como é definida uma interface para criar uma nova fila de requisições. Políticas específicas podem ser implementadas, por exemplo [OMG2000c]:

- Alguns parâmetros de controle podem ser incluídos: máximo número de requisições, tamanho máximo das requisições, tempo máximo que uma requisição pode esperar sem ser processada, prioridades das requisições, etc;
- Estratégias para enfileirar requisições podem ser definidas: FIFO e ordenação por prioridade, ordenação definida pela aplicação através de uma função de *callback* ou *interceptors* (*interceptors* serão abordados na seção 4.3), entre outras;
- Controle de fluxo pode ser introduzido quando a fila está cheia ou quando uma requisição está esperando por muito tempo. Neste caso o cliente pode ser informado ou ainda, as requisições podem ser redirecionadas para outros servidores ou simplesmente descartadas, etc;
- Uma fila de requisições pode ser associada a um estoque de *threads*, de forma que quando uma *thread* ficar pronta ela pode obter automaticamente uma requisição da fila e processa-la.

4.2.5 Transporte

As definições do CORBA padrão buscam a transparência dos meios e protocolos de transmissão. Entretanto, aplicações de tempo real necessitam um controle mais fino sobre a camada de transporte. O RT-CORBA definiu interfaces para criar, remover e especificar os atributos de pontos de comunicação [OMG2000c].

Configuração das propriedades dos protocolos

A comunicação entre cliente e servidor, feita através do protocolo inter-ORB, depende do protocolo de transporte de nível inferior. Por exemplo, o IIOP (*Internet Inter-ORB Protocol*) é implementado pelo GIOP (*General Inter-ORB Protocol*) sobre TCP/IP. Portanto, o IOP é constituído de duas camadas (as quais possuem seus próprios conjuntos de atributos) [SCH2000a]: ORB e transporte.

RT-CORBA define uma interface que permite às aplicações especificar propriedades para os protocolos do ORB e nível de transporte. Os atributos dos protocolos são definidos por uma estrutura *Protocol*, que se reúnem em uma *ProtocolList*, na qual a ordem dos termos é relevante, pois define as preferências da aplicação.

Tanto o cliente quanto o servidor podem definir os atributos para cada um dos protocolos presentes na sua lista de protocolos preferidos. Servidores CORBA podem utilizar a interface *ServerProtocol* (*Policy*) para selecionar quais protocolos (suas preferências) configurar em uma referência para objeto com o intuito de informar aos clientes, inicialmente, quais protocolos estão disponíveis e quais os atributos especificados para os protocolos devem ser utilizados em conexões com o servidor.

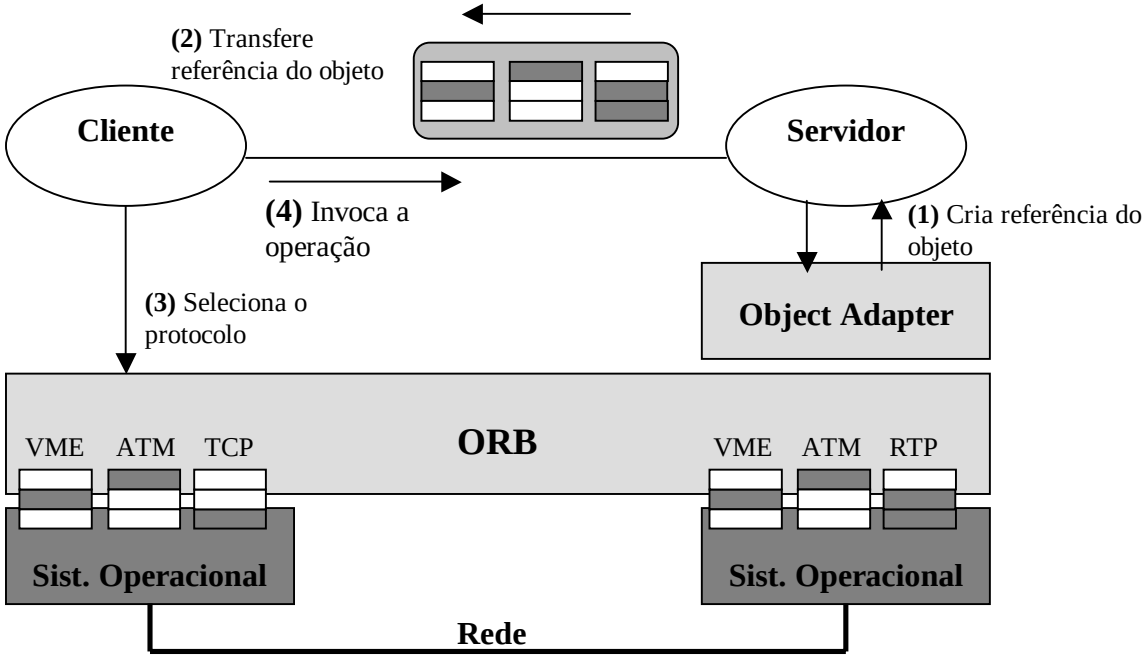


FIGURA 4.5 - Configuração e seleção de protocolos.

Já os clientes podem especificar suas preferências de protocolo através da interface *ClientProtocol* (*Policy*), cujas políticas são aplicadas quando um cliente consegue fazer *biding* para um objeto. A política *ClientProtocol* pode ser especificada tanto pelo cliente quanto pelo servidor, mas apenas por um deles de cada vez. Servidores publicam suas preferências e exigências em um modelo por objeto, já clientes utilizam esta política num modelo por invocação. Quando selecionado pelo servidor a política *ClientProtocol* é propagada para o cliente na referência do objeto.

A FIGURA 4.5 ilustra como um servidor especifica os protocolos disponíveis para o cliente. O servidor publica os protocolos VME, ATM (*Asynchronous Transfer Mode*) e RTP (*Real-Time Transport Protocol*). O cliente então deve concordar com um deles [SCH2000b].

Propriedades em particular de protocolos específicos podem ser definidas através de interfaces herdadas das interfaces genéricas definidas pelo RT-CORBA.

Biding Explícito

O CORBA original suporta apenas *biding* implícito. Neste modelo os recursos utilizados no caminho entre o cliente e o servidor são alocados e as conexões estabelecidas, conforme a demanda, depois da primeira invocação do cliente.

Biding implícito auxilia na transparência de localização e no compartilhamento de recursos de comunicação com diversos servidores. Mas, da mesma forma, este mecanismo é inadequado para aplicações de tempo real e aquelas com exigências de QoS determinísticas, pois pode causar aumento de latência e *jitter*. Ainda, pode ocorrer aumento da inversão de prioridade, porque as filas de conexão são normalmente processadas segundo uma ordem FIFO [SCH2000a].

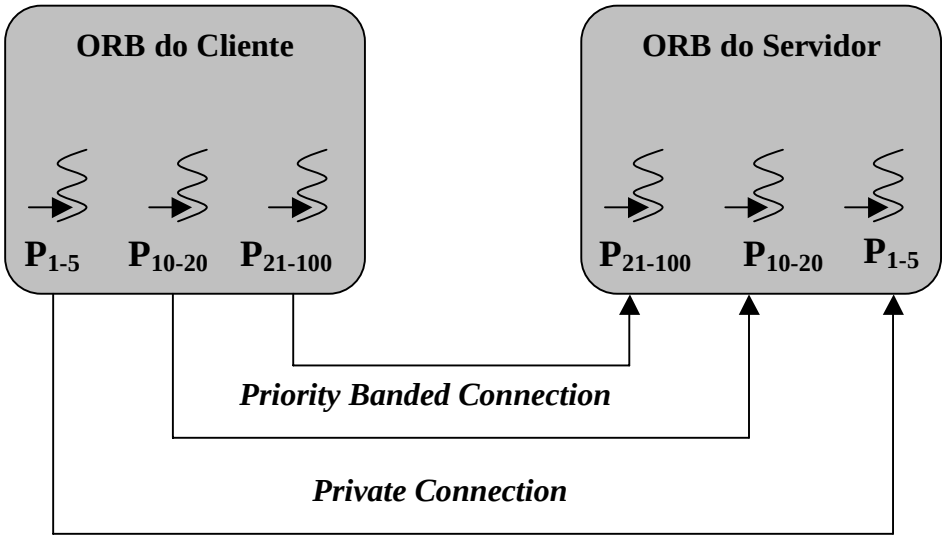


FIGURA 4.6 - Priority Banded Connections e Private Connections.

Para evitar esses problemas, a especificação RT-CORBA define o *biding* explícito, o qual utiliza uma operação de validação da conexão (na especificação do CORBA *Messaging*). Este mecanismo habilita o cliente a pré-estabelecer conexões com os servidores e controlar como as requisições dos clientes são enviadas pelas conexões.

Para suportar o *biding* explícito são definidas duas políticas [SCH2000a] [SCH2000b]:

- ***Priority banded connections*:** permite ao cliente especificar a prioridade de cada conexão de rede e selecionar em tempo de execução uma conexão apropriada de acordo com a prioridade da *thread* que invocou a operação. O cliente é responsável por definir as políticas e envia-las na primeira invocação depois da conexão ter sido criada explicitamente;

- ***Private Connections*:** visto que conexões que são multiplexadas não são muito desejáveis para aplicações de tempo real, o mecanismo de conexões privadas foi criado para permitir ao cliente criar conexões que não sejam compartilhadas por nenhuma outra requisição até que a resposta da primeira tenha chegado.

A FIGURA 4.6 mostra a utilização desses dois mecanismos, onde cada requisição do cliente é enviada por uma conexão pré-alocada, à qual é associada uma prioridade fixa.

4.3 Interceptors

Um ORB de tempo real deve ser flexível o suficiente para permitir modificações ou implementação de novas características para uma invocação de método. Por exemplo, algumas aplicações de tempo real podem precisar passar a prioridade do cliente e utilizar esta prioridade no servidor para implementar herança de prioridade. Outras aplicações podem precisar transmitir informações de QoS mais completas, tais como restrições de tempo, importância e dependências, entre outras. Visto que um ORB não poderia implementar todas as estratégias de tempo real existentes, uma API extensível foi definida para aumentar a flexibilidade do ORB e permitir a criação de novas estratégias. Esta API é definida na forma de interceptadores (*interceptors*).

Um interceptador é um objeto que tem muitas de suas operações chamadas pelo ORB em diferentes níveis de uma invocação. Objetos interceptadores são fornecidos pelas aplicações e podem ser vistos como um tipo especializado de mecanismo de *callback*. Interceptadores são encadeados e suas operações executadas seqüencialmente.

Entre os vários tipos de categorias de interceptadores que podem ser definidas, o RT-CORBA especifica os seguintes [OMG2000c]:

- ***Client Interceptors***: esta categoria representa interceptadores que são chamados pelo ORB quando uma invocação remota é feita. São definidas várias operações que são chamadas em diferentes passos da invocação de métodos. Podem ser utilizados para efeitos históricos, monitoramento ou passagem de informações específicas de QoS.
- ***Server Interceptors***: esta categoria representa interceptadores que são executados pelo ORB quando uma invocação remota é recebida. Várias operações são definidas para tratar cada passo da invocação. Pode ser utilizado para monitorar as atividades do servidor, armazenamento de informações para histórico, bem como receber informações de QoS que podem ser enviadas pelo cliente.
- ***POA Interceptors***: são chamados pelo POA no lado do servidor. Possibilitam a notificação quando o servidor chama o POA para criar uma nova referência a um objeto. São utilizados também para armazenar dados na referência do objeto.
- ***Transport Interceptors***: são utilizados pela camada de transporte do ORB para notificar as aplicações sobre eventos relacionados a esta camada.
- ***Thread Interceptors***: representam interceptadores que são chamados quando o estado de uma *thread* é alterado.

- ***Initialization Interceptors***: são chamados pelo ORB no momento de sua inicialização.
- ***Message Interceptors***: esses interceptadores operam no nível das mensagens. Eles podem ser utilizados para criptografar ou comprimir as mensagens, por exemplo.

A aplicação pode criar vários tipos de objetos interceptadores. A ordem de ativação dos diferentes métodos definidos nos diferentes interceptadores é estabelecida pelo ORB. Por exemplo, quando um cliente faz uma requisição de método, interceptadores no nível de requisição (servidor) são invocados antes dos inteceptadores de mensagens. Quando a resposta é recebida os interceptadores de mensagem são invocados antes dos de resposta (cliente).

Existindo vários interceptadores do mesmo nível (contendo mesmo nome de método e mesmo tipo de objeto interceptador), eles são chamados seqüencialmente. Neste caso, pode-se desejar especificar a ordem com que os métodos são executados e para isso deve-se especificar a prioridade para cada método, visto que os interceptadores são ordenados na lista de execução conforme esse atributo.

É necessário definir duas ordens de chamada de métodos. Por exemplo, se um interceptador criptografa uma mensagem e outro comprime a mensagem quando ela é recebida, deve-se primeiramente descompactar e depois descriptografar a mensagem. Isto é possível somente se forem fornecidos interceptadores com duas ordens semânticas diferentes:

- ***Forward***: os métodos de interceptadores são chamados na ordem de mais alta prioridade. Assim, o método de mais baixa prioridade é chamado por último.
- ***Backward***: por outro lado, há métodos que são chamados segundo a ordem de mais baixa prioridade do interceptador. Neste caso, o método de mais baixa prioridade é chamado primeiro.

Além disso, interceptadores devem seguir algumas regras:

- **Criação**: são criados pelas aplicações, mas uma vez criados eles ainda não estão ativos.
- **Destruição**: podem ser removidos pelas aplicações que os criaram.
- **Ativação e Desativação**: a aplicação que anteriormente criou o interceptador deve ativa-lo explicitamente. No momento da ativação é definida a prioridade do *Interceptor*. ORBs de tempo real podem impor certas restrições para quando a ativação ou desativação pode ser feita, como por exemplo, somente na inicialização.
- **Recursão**: um método de um interceptador pode fazer chamadas ao ORB, bem como fazer uma invocação de método (seja ele remoto ou não). Nestas circunstâncias, dependendo do tipo de operação executada pelo método pode ocorrer de o método interceptador ser chamado novamente pelo ORB. É responsabilidade da aplicação detectar e romper recursões quando necessário.

A interface básica de Interceptor, da qual derivam todos os outros tipos, possui os seguintes elementos básicos: uma enumeração representando o estado da operação do interceptor, a qual pode assumir os valores de INVOKE_CONTINUE (neste caso a invocação pode continuar, e os outros interceptadores na lista são chamados), INVOKE_ABORT (a invocação é abortada e nenhum outro *Interceptor* chamado),

INVOKE_RETRY (é feita uma nova tentativa de execução da invocação); um atributo indicando a prioridade do interceptador; um método de ativação e outro para desativação; e por fim, um método que informa se o *Interceptor* está ativo ou não.

Nas próximas seções, algumas das categorias de interceptadores citadas anteriormente serão detalhadas. Visto que algumas das categorias não tem sua especificação concluída, essas não serão detalhadas.

4.3.1 Request Interceptors

Entre as categorias de interceptadores listadas na seção anterior os *Request Interceptors* representam aqueles que são chamados pelo ORB quando uma invocação é realizada. Envolvem, portanto, um interceptador do lado do cliente e outro no lado do servidor, e ambos devem possuir informações suficientes a respeito da requisição corrente. Abaixo segue a lista de ações mínimas que podem ser tomadas por um interceptador de requisição [OMG2000c]:

- **Obter e definir o objeto alvo da requisição**: obter a referência ao objeto alvo da requisição é importante para extrair informações. Também é importante a possibilidade de altera-lo afim de permitir o roteamento da invocação para outro servidor de forma transparente, possibilitando replicação, tolerância a falhas e outros mecanismos.
- **Obter e definir o nome da operação**: a obtenção do nome da operação pode ser utilizada para aplicações de monitoramento ou efeitos históricos. Também é importante para a troca de informações transparentemente através da passagem de S*ervice Context Data* (dados do contexto do serviço).
- **Observar, remover e disparar exceções**: interceptadores de requisição devem tratar com exceções em vários níveis. Aplicações de monitoramento e efeitos históricos precisam observar as exceções de modo a fazer registros do ocorrido. Mecanismos de tolerância a falhas precisam analisar as exceções de modo a tentar esconde-las (ou seja, remove-las).

Service Context Data

Em alguns momentos é necessário que se troquem informações de forma transparente nas aplicações de tempo real através dos interceptadores. Os dados de contexto do serviço representam esses dados arbitrários que podem ser transmitidos dos interceptadores do cliente para os do servidor durante a invocação de métodos.

É possível especificar vários tipos de dados de contexto. Cada conjunto pode ser independente do outro e cada um pode ser identificado unicamente. Os dados de contexto do serviço são identificados por um identificador de serviço, o qual é representado por um número.

O IIOP suporta a transmissão do contexto do serviço, o qual é um dos campos do pacote do protocolo, nas mensagens de requisição e de resposta. O contexto do serviço pode ser formado por várias estruturas, cada uma contendo um identificador do contexto e os dados desse contexto. Os dados do contexto podem variar conforme o serviço, por exemplo, em determinados casos podem armazenar um número de prioridade.

A interface dos interceptadores define o conjunto básico de operações para tratar com os dados do contexto, tais como: inserir um Service Context Data na mensagem de

requisição ou de resposta; extrair esses dados das mensagens; remover os dados e verificar se um Service Context Data está presente ou não.

Client Interceptors

O interceptador de requisição do lado do cliente possui quatro métodos, os quais são chamados em diferentes passos da invocação. Cada um deles recebe como parâmetro o objeto que descreve a requisição. Apesar da referência do objeto requisitado estar inclusa no objeto que descreve a requisição, nem sempre é possível alterá-la visto que em alguns momentos, como depois da conversão (*marshaling*) da requisição, isto é muito complicado de ser feito.

Os métodos definidos são:

- ***initialize_request***: chamado depois que a referência ao objeto é inicializada, mas antes da criação do cabeçalho da mensagem de requisição no protocolo de interação. A operação é executada pela *thread* do cliente que realizou a invocação.
- ***after_marshal***: chamado depois que o protocolo de interação construiu a mensagem, mas antes do protocolo de transporte ser chamado. A operação também é executada pela *thread* do cliente que efetuou a invocação.
- ***before_unmarshal:*** chamado depois que uma mensagem de resposta é recebida ou quando ocorreu uma falha de comunicação. Em invocações síncronas o método é executado pela *thread* do cliente que efetuou a invocação. Já para invocações assíncronas, o método é executado pela *thread* do *end-point* de transporte.
- ***finish_request:*** chamado quando a invocação termina, de forma correta ou não.

Os métodos *initialize_request* e *after_marshal* seguem a ordem direta de invocação (*forward*), enquanto que os métodos *before_unmarshal* e *finish_request* seguem a ordem inversa (*backward*).

Server Interceptor

Também possui quatro métodos definidos:

- ***initialize_request***: chamado no momento em que uma invocação remota é recebida pelo servidor. O método é executado pela *thread* do *end-point* de transporte.
- ***after_unmarshal***: chamado após os parâmetros da operação terem sido desempacotados do *buffer* de requisição. O método é executado pela *thread* que executará o código da implementação do objeto (*servant*).
- ***before_marshal***: chamado logo depois que a implementação do objeto foi chamada ou quando ocorreu uma exceção. Este é o único método que pode ser utilizado para inserir dados do contexto na resposta. A operação é executada pela mesma *thread* que irá executar o código da implementação do objeto.
- ***finish_request***: chamado quando a invocação termina, ou seja, depois que a resposta é enviada para o cliente.

Os métodos *initialize_request* e *after_unmarshal* seguem a ordem direta (*forward*) de execução, enquanto que os demais seguem a ordem inversa (*backward*).

4.4 Mecanismos de Sincronização

Visto que as especificações do CORBA 2.x não possuem um modelo de *threads*, elas também não possuem formas de assegurar a consistência entre os mecanismos de sincronização da aplicação e dos próprios mecanismos de sincronização internos ao ORB. Entretanto, esse tipo de sincronização é necessário, não somente em aplicações de tempo real, mas mais genericamente em aplicações *multi-thread* [SCH2000a] [SCH2000b].

Desta forma, visto que objetos de sincronização são ferramentas muito importantes e com o desejo de aumentar a portabilidade de aplicações de tempo real, o RT-CORBA define uma API para os seguintes objetos:

- *Mutex*
- Semáforo
- Múltiplos Leitores/Único Escritor
- Variáveis de condição

Mutex

A interface *Mutex* oferece operações padronizadas para garantir a exclusão mútua, são elas:

- ***acquire***: adquire o *mutex*. Caso o *mutex* esteja liberado, a *thread* pode entrar na sessão crítica. Caso contrário, ela será bloqueada até isso acontecer.
- ***release:*** libera o *mutex*. Caso alguma *thread* esteja bloqueada nesse *mutex*, uma delas será liberada.
- ***try_acquire:*** método que tenta adquirir o mutex. Caso ele esteja livre, será retornada uma resposta verdadeira para o método e o *mutex* será adquirido. Se, pelo contrário, o *mutex* não estiver livre, o método retornará um resultado falso e a *thread* não será bloqueada.
- ***destroy:*** destrói o objeto *Mutex*.

Semáforo

A interface *Semaphore* oferece operações padronizadas para controlar um contador do tipo semáforo, o qual é um inteiro normalmente utilizado para gerenciar recursos. As operações especificadas são:

- ***acquire***: decrementa o contador do semáforo. Se o contador chegar a um valor negativo, a *thread* que invocou o método será bloqueada.
- ***try_acquire:*** operação similar ao método *acquire*, com exceção de que retorna verdadeiro caso tenha decrementado o valor com sucesso e falso em caso contrário, sem bloquear a *thread* que chamou o método.

- ***release:*** método que incrementa o valor do semáforo. Se o novo valor é positivo ou zero, uma *thread* que havia sido bloqueada pelo semáforo será acordada.
- ***destroy:*** destrói o objeto semáforo.

Múltiplos Leitores/Único Escritor

Muitas *threads* podem ter acesso simultâneo a dados para leitura, enquanto somente uma *thread* pode ter acesso de escrita em um dado momento. A utilização de outros mecanismos de sincronização comuns poderia causar contenção e diminuição do paralelismo. Dessa forma, o RT-CORBA define um mecanismo de sincronização especial, a interface *RWLock*, a qual serializa o acesso aos recursos que são mais consultados do que modificados.

Assim a interface *RWLock* define as seguintes operações:

- ***acquire_write*****:** método que adquire o *lock* para escrita se não houver nenhum outro escritor e leitores. Caso o *lock* não esteja livre, a *thread* será bloqueada até que não existam mais leitores nem escritores.
- ***try_acquire_write***: operação idêntica a *acquire_write*, exceto que não bloqueia a *thread* que invocou a operação. Ela retorna verdadeiro caso o escritor tenha conseguido o *lock* e falso em caso contrário.
- ***acquire_read:*** operação que tenta adquirir o *lock* para leitura. Caso exista algum escritor com o *lock*, a *thread* será bloqueada até que o escritor libere-o.
- ***try_acquire_read:*** identifica a operação *acquire_read*, exceto que ela não bloqueia a *thread* que invocou a operação, apenas retorna verdadeiro quando conseguiu obter o *lock* e falso em caso contrário.
- ***release:*** operação que libera o *lock*. Caso existam *threads* bloqueadas pelo *lock*, seja de leitura ou escrita, uma delas será acordada.
- ***destroy:*** destrói o objeto *RWLock*.

Variável de condição

Uma variável de condição habilita *threads* a bloquear atomicamente e testar a condição sob proteção de um bloqueio de exclusão mútua (*Mutex*) até que a condição seja satisfeita. A interface *ConditionVariable* oferece as seguintes operações para controlar e utilizar variáveis de condição:

- ***wait:*** recebe como parâmetro um objeto do tipo *Mutex* e um tempo máximo de espera. Este método atomicamente libera o *Mutex* e bloqueia a *thread* chamada na variável de condição (até o tempo máximo de espera indicado) até que outra *thread* invoque uma operação *signal* ou broadcast. Retorna verdadeiro se a variável de condição foi sinalizada ou falso no caso do tempo limite de espera ter terminado.
- ***signal:*** este método desbloqueia uma *thread* que estava esperando na variável de condição.
- ***broadcast:*** esta operação desbloqueia todas as *threads* que estavam esperando na variável de condição, e todas se tornam prontas para executar.
- ***destroy:*** destrói o objeto *ConditionVariable*.

4.5 Serviço de Escalonamento do RT-CORBA

O Serviço de Escalonamento utiliza primitivas do RT-ORB para garantir as várias políticas de escalonamento baseadas em prioridade fixa do RT-CORBA de uma forma a abstrair das aplicações algumas das construções de tempo real. O Serviço de Escalonamento não impõe nenhuma nova exigência aos ORBs e RT-ORBs além daquelas que aparecem nas especificações do CORBA e RT-CORBA respectivamente.

As primitivas adicionadas no RT-CORBA para criar o RT-ORB são suficientes para se obter um escalonamento de tempo real, mas um escalonamento desse tipo, efetivamente, é complicado. Para aplicações assegurarem que sua execução é escalonável de acordo com uma política uniforme, tal como Taxa Monotônica (*Rate Monotonic*) global, exige-se que as primitivas do RT-ORB sejam utilizadas apropriadamente e que seus parâmetros sejam atribuídos de forma correta em todas as partes do sistema CORBA [OMG2000e].

Não é somente a determinação dos parâmetros corretos uma atividade difícil, mas também o próprio código fonte da aplicação se torna substancialmente mais complexo, tornando a análise e modificações muito difíceis. O Serviço de Escalonamento procura encapsular muito dessa complexidade abstraindo detalhes da política de escalonamento do código da aplicação.

Assume-se que se uma aplicação utiliza o Serviço de Escalonamento ela possui uma política de escalonamento uniforme, e o Serviço de Escalonamento escolherá as prioridades CORBA, as políticas do POA e os mapeamentos de prioridade de forma a garantir isso. Entretanto, diferentes implementações do Serviço de Escalonamento podem oferecer diferentes políticas de escalonamento de tempo real.

A abstração utilizada no Serviço de Escalonamento para os parâmetros de escalonamento (tais como prioridades CORBA) é a utilização de nomes. No código da aplicação são utilizados nomes (*strings*) para especificar atividades ou objetos CORBA. O Serviço de Escalonamento internamente associa os nomes com os parâmetros e políticas de escalonamento para aqueles objetos ou atividades. Esta abordagem melhora a portabilidade em relação às funcionalidades de tempo real, facilita a utilização dessas funcionalidades e reduz a chance de erros [OMG2000e].

Cada nome utilizado no Serviço de Escalonamento deve ser único. O Serviço de Escalonamento foi projetado para trabalhar em um sistema CORBA fechado, onde são utilizadas prioridades fixas para um conjunto também fixo de clientes e servidores. Então, assume-se que o projetista do sistema identificou um conjunto estático de atividades CORBA, objetos CORBA que essas atividades utilizam, e para estes objetos e atividades determinou parâmetros de escalonamento, tais como a prioridade. Neste processo os nomes são unicamente associados àquelas atividades e objetos e esses nomes finalmente associados aos parâmetros de escalonamento. Esta associação é então utilizada para configurar o Serviço de Escalonamento.

Utilizando o Serviço de Escalonamento, as aplicações não precisam fazer invocações diretamente às APIs de programação do CORBA de tempo real, tais como o RT-ORB ou o RT-POA. Os detalhes referentes ao escalonamento dos objetos e atividades são deixados para o Serviço de Escalonamento. Assim como pode ser visto na TABELA 4.1, tanto o cliente quanto o servidor, dispõe de uma interface de programação simples. Assim que o cliente indica uma atividade a ser escalonada (ou no

caso do servidor um objeto), a responsabilidade de atribuir a prioridade correta (segundo o *PriorityMapping*) é deixada para o Serviço de Escalonamento. Alterações no modelo de prioridades podem ser feitas, portanto, diretamente no serviço sem que uma recompilação da aplicação seja necessária.

TABELA 4.1 - Definição da interface do Serviço de Escalonamento em IDL.

Módulo/Interface/Método	Explanação
`module RTScheduling{`	Módulo que contém as interfaces pertencentes ao Serviço de Escalonamento.
`exception UnknownName{};`	Exceção definida para quando é utilizado um nome de objeto ou atividade não válido em algum dos métodos das interfaces do módulo.
`interface ClientScheduler{`	Interface do Serviço de Escalonamento utilizada pelo cliente.
`void schedule_ativity( in string name) raise(UnknownName)`	O cliente invoca esse método sempre que entra em uma região de código que possui um novo deadline ou prioridade (indicando uma nova atividade CORBA), informando assim o nome da nova atividade a entrar no Serviço de Escalonamento, o qual associa uma prioridade à atividade e é responsável por invocar as primitivas adequadas do RT-ORB ou do RT-OS.
`};`	
`interface ServerScheduler{`	Interface do Serviço de Escalonamento utilizada pelo servidor.
`POA create_POA( in POA parent, in string adapter_name, in POAManager a_POAManager, in PolicyList policies) raises(AdapterAlreadyExists, InvalidPolicy);`	O POA criado por este método garantirá todas as políticas (não de tempo real) indicadas pelo parâmetro *policies*. Enquanto que todas as políticas de tempo real serão asseguradas internamente pelo método, de modo a serem consistentes com as políticas adotadas pela implementação do Serviço de Escalonamento utilizado.
`void schedule_object( in Object obj, in string name) raises(UnknownName);`	Este método permite o controle de escalonamento em nível de objeto. Os parâmetros necessários para o escalonamento, assim como a prioridade, por exemplo, serão derivados internamente pelo Serviço de Escalonamento, utilizando para isso o nome indicado como parâmetro.
`};`	
`};`	

4.6 Estudo de caso: TAO

Até este ponto foram analisadas as especificações do CORBA e do RT-CORBA, respectivamente. Visto que a OMG não tem como função criar nem definir aspectos

mais específicos de uma implementação CORBA (seja de tempo real ou não) é importante que se faça uma breve análise de alguma das implementações disponíveis no mercado.

Já estão disponíveis várias implementações de CORBA tanto de forma comercial quando implementações livres (*free software*). Entretanto, o número de implementações do RT-CORBA não é tão grande. Além disso, as implementações existentes muitas vezes não seguem exatamente as especificações existentes e, mais comumente ainda, acrescentam características não definidas na recomendação da OMG.

Dentre as implementações de RT-CORBA destaca-se o TAO, *"The ACE ORB"*. Dentre as deficiências do CORBA para aplicações de tempo real, o TAO preocupou-se fundamentalmente com a especificação e garantia de QoS, escalonamento de tarefas de tempo real e com a performance do ORB. Desta forma, o TAO é mais ajustado para aplicações de tempo real, tanto *hard* quanto *soft*, em um ambiente estático [FEN2000].

O TAO foi escolhido como plataforma para este trabalho principalmente por ser o mais completo e estável dos ORBs de tempo real encontrados e por ser totalmente livre tanto para fins comerciais quanto acadêmicos. Apesar da ausência de uma documentação mais consistente[1] muitos artigos científicos e uma lista de discussão (*tao-users@cs.wustl.edu*) bastante ativa auxiliam na compreensão das funcionalidades do TAO e solução de eventuais problemas durante a sua instalação e utilização.

Nas próximas seções deste capítulo serão vistos os principais componentes da arquitetura do TAO.

4.6.1 Visão Geral

O TAO (*The ACE ORB*) é um ORB para RT-CORBA que está sendo desenvolvido no *Distributed Object Computing (DOC) Group*, o qual é uma parceria entre o *Center for Distributed Object Computing – Computer Science Department – Washington University, St. Louis* e o *Laboratory for Distributed Object Computing – Eletrical and Computer Engineering Department – University of California, Irvine* – Estados Unidos. Pode-se considerar o TAO mais do que apenas um ORB de comunicação de tempo real, mas sim uma arquitetura de um sistema final para execução de aplicações de tempo real, tanto *hard* quanto *soft*.

A arquitetura do TAO contém as seguintes características e elementos [SCH2000c]:

1. **Subsistema de E/S de tempo real**: otimiza os sistemas de E/S dos sistemas operacionais convencionais de forma a oferecer processamento de protocolos de forma previsível a taxas na casa dos gigabits através de redes ATM.
2. **ORB**: fornece um sistema para entrega de requisições dos clientes para o *Object Adapter* e retorno de respostas flexível, portável e eficiente.
3. **Protocolo GIOP de tempo real**: transmite de forma eficiente e previsível requisições utilizando avanços do protocolo de interoperabilidade CORBA.
4. **Serviço de escalonamento de tempo real**: determina a prioridade com a qual as requisições dos clientes são despachadas pelo *Object Adapter*.

[1] Exemplos de código são incluídos no pacote gratuito do ACE/TAO, mas manuais propriamente ditos não estão disponíveis de forma gratuita, foram produzidos e são comercializados por uma empresa de consultoria, a OCI – Object Computing Inc. (http://www.theaceorb.com/).

5. ***Object Adapter* de tempo real**: demultiplexa e despacha requisições de clientes de acordo com os requisitos de tempo real determinados pelo serviço de escalonamento.
6. **Compilador IDL e camada de apresentação**: são otimizadas as fontes principais de sobrecarga na codificação e decodificação de requisições.
7. **Otimizações no gerenciamento de memória**: minimiza as fontes de alocação dinâmica de memória e cópia de dados através das camadas do ORB.
8. **Especificação de QoS**: permite às aplicações e aos serviços CORBA de mais alto nível definir parâmetros de QoS fim-a-fim utilizando o modelo de programação orientado a objetos.

Esses elementos que compõe a arquitetura do TAO podem ser vistos na FIGURA 4.7.

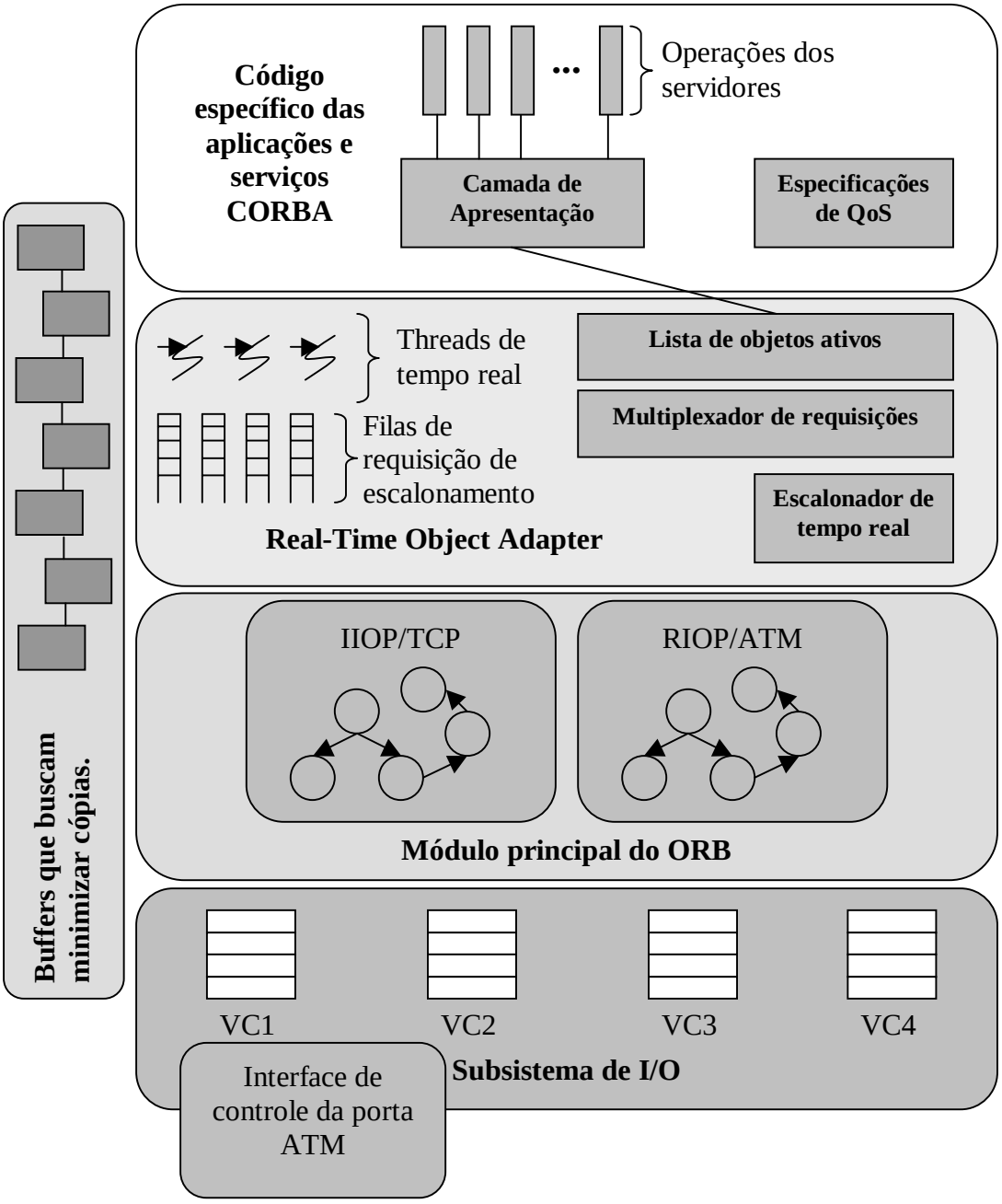


FIGURA 4.7 - Elementos da arquitetura do ORB TAO.

Nas próximas seções serão detalhados os elementos que compõem a arquitetura do ORB TAO, apresentando suas principais características.

4.6.2 Subsistema de E/S

O subsistema de E/S é responsável por mediar o acesso do ORB e das aplicações ao nível de rede e recursos do sistema operacional como controladores de dispositivos, pilhas de protocolo e processador (es). Os principais pontos no desenvolvimento de um sistema desse tipo são: garantir os requisitos de QoS enquanto diminui-se o não-determinismo e a inversão de prioridade, tornar conveniente às aplicações a forma de especificar os requisitos de QoS e habilitar o ORB a oferecer as garantias de QoS fornecidas pela rede.

De modo a alcançar estes objetivos, foi desenvolvido um subsistema de rede para E/S de alto desempenho. Os componentes neste subsistema incluem [SCH2000d]:

- **Adaptador de rede de alta velocidade:** Um dos pontos importantes do sistema de E/S desenvolvido para completar o sistema do TAO é um controlador de interconexão ATM (*ATM Port Interconnection Controller* – APIC, *hardware*, não precisa estar presente para o TAO funcionar). Pode ser utilizado como uma interface de rede ou como um chip de interface de E/S. Pode sustentar uma taxa bi-direcional de 2.4 Gbps. Além disso, como o TAO possui uma arquitetura baseada em camadas ele pode rodar em sistemas convencionais interligados por conexões de tempo real.
- **Subsistema de E/S de tempo real:** Alguns sistemas operacionais (tais como *Solaris* ou *Windows NT*) suportam escalonamento de tempo real. Entretanto, sistemas operacionais de propósito geral não oferecem um subsistema de E/S de tempo real. O TAO faz alterações no modelo STREAMS[2] da plataforma *Solaris* de forma a superar suas limitações no que se referem ao não suporte a requisitos de QoS e inversão de prioridade visto que chamadas ao sistema de E/S são executadas em uma prioridade menor que as demais tarefas de tempo real (e assumem um valor único de prioridade, indiferente ao processo que está fazendo E/S). As alterações efetuadas seguem principalmente a estratégia de adotar um conjunto de *kernel threads* associadas as *threads* da aplicação. As *kernel threads* executam na mesma prioridade que as *threads* da aplicação, evitando assim a inversão de prioridade [SCH2000e].
- **Escalonador de tempo real:** TAO suporta garantias de QoS através de um escalonador de E/S que suporta aplicações periódicas de tempo real. Assim que uma thread da classe de E/S de tempo real é admitida pelo sistema operacional, o escalonador é responsável por atribuir sua prioridade em relação a outras *threads* da classe e despachar a *thread* periodicamente de modo que seu *deadline* seja cumprido. A atual implementação do TAO permite às aplicações especificarem seus requisitos de uma forma intuitiva para o escalonador de E/S. Isto é feito em termos no tempo de computação (C) e do

[2] O subsistema de comunicação do sistema operacional *Solaris* é baseado no modelo STREAMS, o qual fornece um modelo de implementação com uma variedade de componentes reutilizáveis, tais como gerenciamento de *buffers*, módulos de composição de protocolos em camadas e controle de fluxo camada-a-camada. As camadas de protocolo podem ser *multi-thread*, sendo que no sistema original as *threads* executam utilizando um valor de prioridade do sistema (classe SYS) o qual é inferior às prioridades das tarefas de tempo real.

período da tarefa (P), o escalonador atribui a prioridade a *thread* de E/S de forma que sua escalonabilidade seja garantida.

- **Controlador de admissão:** Para assegurar que os requisitos de QoS da aplicação sejam alcançados, TAO efetua um controle de admissão para o escalonador de E/S. Este permite que o sistema operacional garanta o tempo de computação especificado ou então recuse-se a admitir a *thread*. Este mecanismo é útil tanto para sistemas de tempo real determinísticos quanto para estatísticos.

4.6.3 Módulo principal do ORB

O módulo principal de um ORB fornece um sistema em tempo de execução que gerencia conexões de transporte, envia requisições dos clientes para o *Object Adapter* e retorna repostas aos clientes (quando elas existem). O desafio em um ORB de tempo real é projetar um ORB eficiente que possa ser estendido facilmente para se adaptar a novos ambientes e exigências de aplicações.

De modo a aumentar sua facilidade de extensão o módulo principal do ORB TAO é baseado no *framework* ACE[3] [SYY2000]. Os modelos de *Acceptor* e *Connector* são utilizados no TAO para dividir a tarefa de estabelecer conexões e efetuar os procedimentos que ocorrem depois que as conexões são estabelecidas. Ainda, utiliza o modelo *Reactor*, o qual simplifica as partes do ORB que são orientadas a eventos integrando demultiplexação e despacho dos identificadores de eventos. Além destes, também é utilizado o modelo de *Active Object* para simplificar a concorrência de acesso aos componentes do ORB através da dissociação entre a invocação dos métodos e a execução dos mesmos.

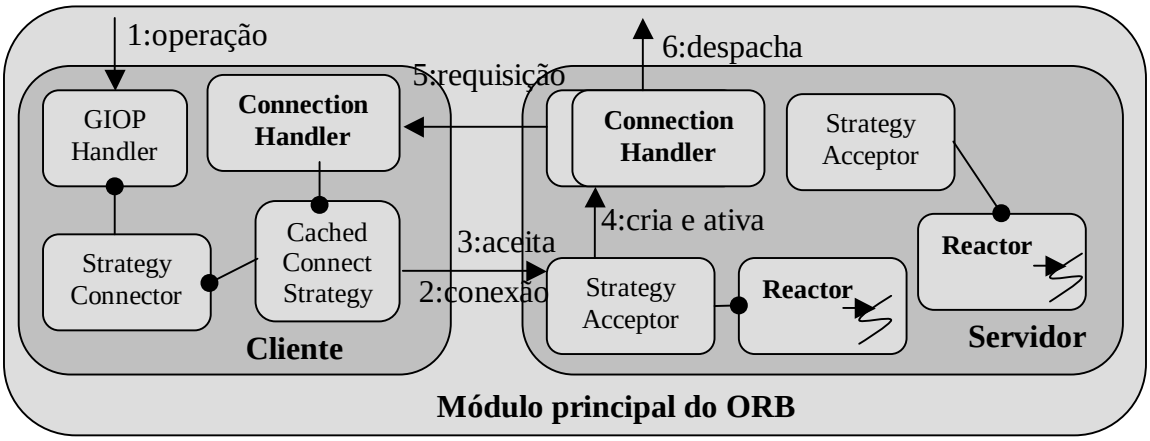


FIGURA 4.8 - Componentes do módulo principal do ORB TAO.

[3] O ambiente de comunicação adaptativo (*Adaptive Communication Environment* – ACE) é um *framework* (e um conjunto de ferramentas) orientado a objetos que implementa modelos fundamentais de concorrência e distribuição para software de comunicação. O ACE inclui vários componentes para auxiliar no desenvolvimento de software de comunicação e para alcançar flexibilidade, eficiência, confiabilidade e portabilidade. Sua arquitetura é baseada em várias camadas (camada de adaptação ao sistema operacional, camada de empacotamento em C++, camada de modelos e ferramentas). Pode ser utilizado para vários propósitos como: comunicação e concorrência entre processos, gerenciamento de memória, temporizadores, sinalizadores, gerência de *threads*, construção de protocolos em camadas, estabelecimento de conexões e inicialização de serviços, serviços de comunicação distribuída entre outros.

A FIGURA 4.8 ilustra os componentes do módulo principal do ORB TAO tanto no lado do cliente quanto do lado do servidor. No cliente é utilizado um ***Strategy_Connector*** para armazenar de forma temporária conexões com o servidor salvando, dessa forma, as configurações das conexões e conseguido uma diminuição na latência entre invocação e execução. No lado do servidor, o ***Reactor*** detecta as conexões que chegam e notifica o ***Strategy_Acceptor***. Este, por sua vez, aceita a nova conexão e associa esta com um ***Connection_Handler*** que executa em uma *thread* com a prioridade de tempo real apropriada. O ***Connection_Handler*** do cliente passa a requisição através do RIOP (*Real-time Inter-ORB Protocol*) para o ***Connection_Handler*** do servidor, o qual faz a chamada para o *Object Adapter* de tempo real para escalonar e despachar a requisição para a implementação do objeto (*servant*).

4.6.4 Protocolo GIOP de tempo real

CORBA foi projetado para executar sobre múltiplos protocolos de transporte. O protocolo padrão de interoperabilidade, o GIOP, normalmente é implementado com o IIOP, o qual roda sobre TCP/IP.

Entretanto, um ORB não está limitado a rodar apenas sobre TCP/IP que, apesar de oferecer confiabilidade, controle de fluxo e demais funcionalidades ao IIOP, pode não ser o mais indicado para aplicações de tempo real. Da mesma forma, quando clientes e servidores estão na mesma máquina, um esquema de comunicação baseado em memória compartilhada pode ser mais eficiente. Assim, um ponto chave no projeto de um ORB é ser flexível o suficiente para suportar múltiplos mecanismos de comunicação.

Além de suportar um conjunto grande de protocolos de transporte, o ORB TAO oferece uma implementação otimizada do IIOP. Ainda, é fornecido um protocolo *Inter-ORB* de tempo real (*Real Time Inter-ORB Protocol* – RIOP) que estende o GIOP/IIOP com atributos de QoS [SCH2000c].

Real Time Inter-ORB Protocolo (RIOP)

Visto que nem o GIOP nem o IIOP oferecem qualquer suporte à especificação e garantia das exigências de QoS fim-a-fim, as aplicações de tempo real não podem suportar a latência e ocorrência de *jitter* presente nesses protocolos em decorrência da utilização do TCP/IP. Da mesma forma, protocolos de roteamento como IPv4 não possuem funcionalidades de controle de admissão de pacotes e controle da taxa de transferência, o que pode causar congestionamentos excessivos e perda dos deadlines nos sistemas de comunicação.

Assim, para melhorar a previsibilidade e o desempenho, TAO suporta o RIOP (*Real Time Inter-ORB Protocol*), o qual é um mapeamento do GIOP que permite às aplicações transferir seus parâmetros de QoS fim-a-fim entre clientes e servidores. As informações de QoS são transferida no campo *service_context*, parte integrante do cabeçalho das mensagens de requisição do GIOP. Desta forma, é assegurada a compatibilidade com implementações do IIOP existentes que não suportem as extensões de QoS do TAO, as quais podem ignorar o campo *service_context*.

O *service_context* correspondente a cada invocação de um cliente e pode conter atributos como: prioridade, período de execução e classe de comunicação. No que se refere às classes de comunicação suportadas pelo TAO, podem-se citar:

- ***ISOCHRONOUS***: para mídia contínua.
- ***MESSAGE***: para mensagens de tamanho pequeno com exigências de pouco atraso.
- ***MESSAGE_STREAM***: utilizada para seqüências de mensagens que podem ser processadas em uma certa taxa.

Além de transportar atributos de QoS, RIOP é projetado para mapear o CORBA GIOP para uma variedade de redes (incluindo redes de alta velocidade como ATM LANs e ATM/IP WANs) e pode ser estendido para exigências específicas de aplicações. Por exemplo, em aplicações que não necessitam de uma completa confiabilidade (tal como aplicações de vídeo conferência), o RIOP pode omitir a funcionalidade da camada de transporte (como detecção de erro em nível de bit, retransmissão, etc.) e rodar diretamente sobre os circuitos virtuais do ATM.

A implementação do RIOP presente no TAO é uma versão otimizada para tempo real da implementação do IIOP da *SunSoft* que é integrada com o sub-sistema de E/S de alto desempenho descrito na seção 4.6.2. Assim, o ORB no cliente, no servidor, e em qualquer nodo intermediário pode colaborar com o processamento das requisições RIOP de acordo com seus atributos de QoS. Este permite aos clientes indicar as prioridades relativas das suas requisições de modo que o TAO possa assegurar suas exigências de QoS (tal como a taxa em que operações periódicas devem ser executadas).

Ainda, de modo a aumentar a portabilidade através do sistema operacional e plataforma de rede, a implementação do RIOP no TAO foi projetada como uma camada separada no módulo principal do ORB. Desta forma, ele pode ser tanto integrado com o subsistema de E/S do próprio TAO, quanto a uma plataforma de comunicação de sistemas de tempo real embarcados.

4.6.5 Serviço de Escalonamento de Tempo Real

O Serviço de Escalonamento do TAO é responsável por alocar os recursos do sistema para garantir as necessidades das aplicações. Sendo mais direcionado para aplicações de *hard real time*, seu principal objetivo é garantir que as exigências de recursos sejam asseguradas. É dividido em componentes de tempo de projeto (*offline*) e de tempo de execução (*runtime*). Na primeira é analisada a possibilidade de escalonamento correto de todas as operações e são atribuídas as suas prioridades. Os componentes de tempo de execução oferecem acesso rápido aos valores de prioridades e coordena as mudanças de modos.

A FIGURA 4.9 apresenta uma visão de mais alto nível dos componentes *offline* e *runtime* do Serviço de Escalonamento. A aplicação registra uma *RT_Info* para cada tarefa do sistema e também registra as dependências entre as operações registradas. Essas dependências podem ser *two-way* (quando há uma invocação de método bloqueante, como um método que possui retorno) ou *one-way* (quando a invocação não é bloqueante, é o caso dos métodos *oneway* definidos em IDL). A partir destas informações o escalonador o*ffline* constrói as seqüências de operações (representadas pelas estruturas *RT_Operations*), é calculada a possibilidade de escalonamento do sistema e são geradas as tabelas de prioridade.

O Serviço de Escalonamento garante que todas as *threads* no sistema terão tempo suficiente para cumprir os seus *deadlines*. Para alcançar isso, o Serviço de Escalonamento pode ser implementado para executar várias políticas de escalonamento

de tempo real. A política adotada inicialmente pelo TAO é utilizando Escalonamento *Rate Monotonic* (Taxa Monotônica) [SCH2000d]. Na política de *Rate Monotonic* prioridades mais altas são atribuídas a tarefas com maior freqüência de execução. Para ser possível realizar uma análise *Rate Monotonic* (e dessa forma executar o teste para verificar se é possível a geração de um escalonamento válido) as aplicações devem fornecer dados de tempo de computação e período para cada tarefa. Além disso, o escalonador supõe inicialmente que a aplicação segue as regras abaixo, relaxando-as progressivamente (o que dificulta o processo de encontrar uma escala possível, mas torna mais viável para o programador adequar o sistema às exigências do escalonador) até que encontre uma forma na qual possa ser aplicada a análise de taxa monotônica sobre as tarefas cadastradas.

- Todas as tarefas são independentes;
- Todas as tarefas são periódicas;
- Não há interrupções;
- Tempo de troca de contexto pode ser ignorado;
- Existe apenas um nível de prioridades;
- Existe apenas um processador;
- Períodos das tarefas são relacionados de forma harmônica;
- Todos os *deadlines* das tarefas são no final dos períodos;

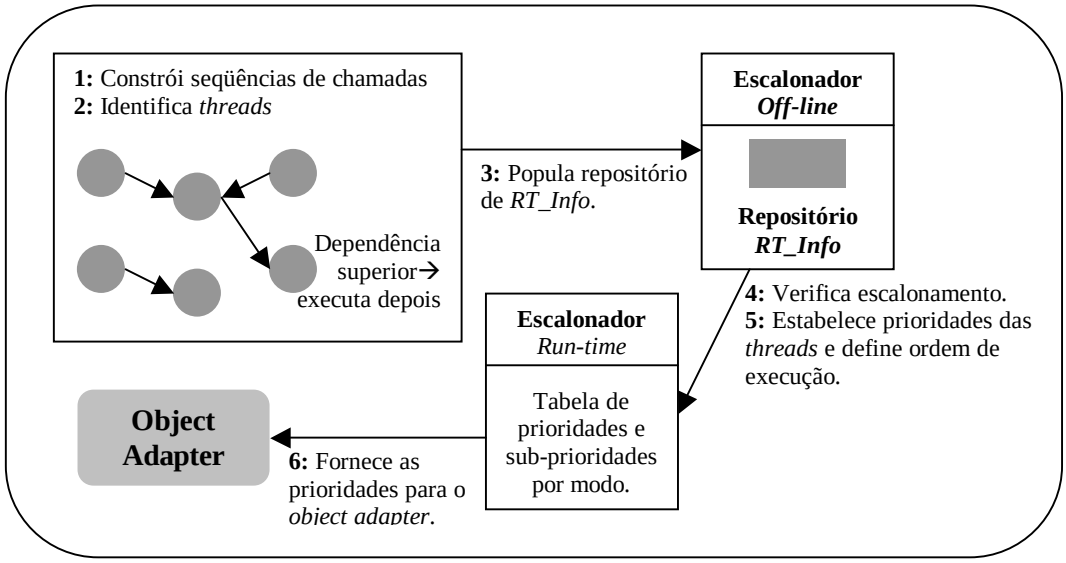


FIGURA 4.9 - Componentes do Serviço de Escalonamento do TAO.

Escalonador *Run-Time*

Os componentes do Escalonador em Tempo de Execução do TAO são os seguintes:

- ***Work_Task***: unidade de trabalho que encapsula atividades de comunicação e processamento do nível da aplicação. Em sistemas de tempo real, uma *Work_Task* normalmente é denominada de módulo ou processo.

- ***RT_Task***: é uma *Work_Task* que possui restrições de tempo. Cada estrutura dessas é considerada uma operação (ou seja, um método definido em um objeto alvo), a qual possui suas próprias informações de QoS em termos de atributos no seu descritor de informações de tempo de execução (*RT_Info*).
- ***Thread***: são as unidades de execução concorrente. Todas as *threads* estão contidas em estruturas *RT_Task*. Quando uma *RT_Task* contém uma ou mais *threads* é chamada de "objeto ativo" e quando não contém nenhuma é chamado de "objeto passivo", os quais são executados no contexto de outro *RT_Task* que fez a invocação da operação.
- ***OS Dispatcher***: utiliza as prioridades das *threads* para selecionar a próxima que assumirá o processador. A *thread* que está no processador é retirada quando não mais em execução ou quando outra de maior prioridade está pronta para execução. Na política adotada de *Rate Monotonic* são adotadas prioridades fixas, portanto o sistema operacional não altera a prioridade das *threads*.
- ***RT_Info***: é a estrutura que especifica as características de escalonamento (tais como tempo de computação e período de execução) de uma *RT_Task*.
- ***Escalonador em tempo de execução***: é o elemento visível do Serviço de Escalonamento em tempo de execução. Informa prioridades ao *Object Adapter* em resposta a requisições de tarefas específicas.

Na implementação do Serviço de Escalonamento do TAO é utilizada a política *Rate Monotonic* com prioridades fixas assim, qualquer alteração que se necessite fazer em termos de parâmetros do escalonamento obtém-se resultado apenas com alteração das prioridades das *threads*. Dessa forma, foi criado o conceito de modos, o qual pode ser interpretado como um regime operacional definido pelo seu conjunto de *threads* e seus relativos parâmetros de escalonamento. Alterações nas *threads* que compõe o sistema, ou em seus atributos, implicam na alteração do modo corrente do sistema.

Os passos necessários para uma mudança de modo são os seguintes:

- A *thread* do escalonador de tempo de execução é acordada e toma o processador;
- O escalonador em tempo de execução determina e armazena uma indicação do novo modo a ser adotado;
- O escalonador em tempo de execução atualiza suas tabelas de prioridades para o novo modo;
- O escalonador em tempo de execução notifica todas as *threads* de infra-estrutura e de aplicação sobre o novo modo através de notificações de forma assíncrona (através de um sinal) ou através de uma abordagem na qual as *threads* devem cooperativamente preemptar-se umas às outras;
- O escalonador em tempo de execução pode voltar a dormir.

Um dos problemas da utilização de modos é a possível demora na notificação assíncrona ou cooperação entre as *threads* para que todas elas tomem ciência do novo modo adotado.

Escalonador *Off-line*

O escalonador *off-line* tem o objetivo de suprir as exigências dos sistemas de tempo real de escalonamento estático. Possui fundamentalmente dois objetivos: (i)

analisar a possibilidade de escalonamento do sistema, reportando antes do momento de execução do sistema quando um ou mais *deadlines* do conjunto de *threads* do sistema pode ser perdido; (ii) associar prioridades às *threads* quando um esquema de escalonamento possível é encontrado.

A análise sobre a viabilidade de escalonamento das tarefas cadastradas pela aplicação no Serviço de Escalonamento do TAO é feita utilizando, atualmente, uma política *Rate Monotonic*, através das informações de tempo de processamento e período das tarefas contidas na estrutura *RT_Info*. Através dessas informações básicas das cadeias de dependências (formadas por estruturas *RT*_Task) entre as tarefas e do cálculo da utilização exigida e dos recursos disponíveis conhecidos, o Serviço de Escalonamento *Off-line* informa a aplicação quando o escalonamento das tarefas em questão não é possível, ou então inicia o processo de geração das tabelas de prioridades.

Ainda, devido a utilização da política de *Rate Monotonic*, cada *RT_Task* recebe uma prioridade de acordo com a sua periodicidade. Tarefas de menor período recebem prioridades mais altas, e vice-versa. Tarefas com período zero (isto é, que não possuem exigências de freqüência na sua execução), recebem sua prioridade de acordo com uma análise da lista de dependências presente na estrutura *RT_Info* (é utilizado o menor período encontrado nessa lista para se gerar a prioridade para a tarefa em questão). A prioridade então calculada para a *RT_Task* é armazenada no campo *RT_Info::priority* para que possa ser acessada através de uma operação CORBA pelo Escalonardor *Run-time*.

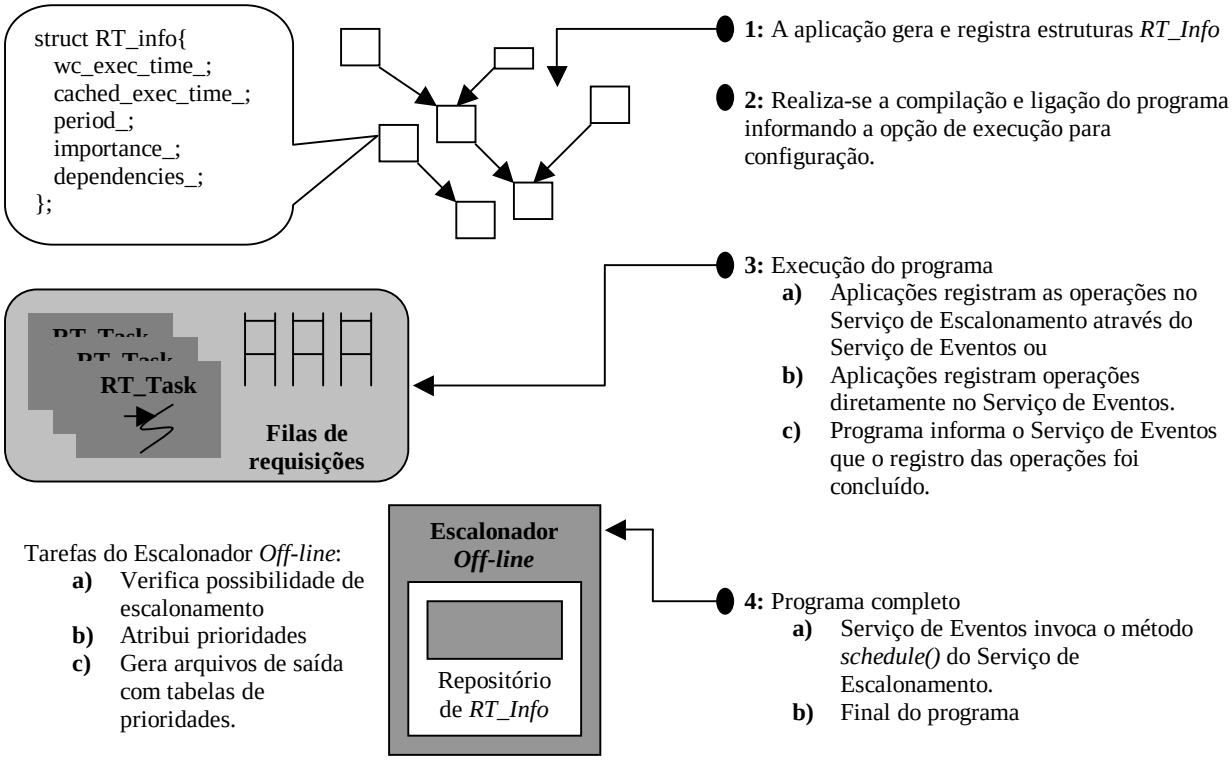


FIGURA 4.10 - Passos executados durante uma execução de configuração.

As fases de determinação da possibilidade de escalonamento, da atribuição de prioridades para as tarefas e posterior geração das tabelas de prioridade para cada modo são precedidas por outras duas fases de igual importância: (i) a extração das estruturas *RT_Info* e (ii) identificação das *threads* de tempo real.

O Serviço de Escalonamento é um objeto CORBA que pode ser acessado pelas aplicações durante a execução de sua configuração (é uma execução da aplicação, mas ainda não em seu modo normal). O usuário deve fornecer instanciações de estruturas *RT_Info* para cada ponto de entrada, e para cada modo de operação do sistema. Essas instanciações de *RT_Info* são compiladas e ligadas no programa principal da aplicação. Os passos que as aplicações e os componentes de infra-estrutura devem fazer para o escalonamento de suas tarefas é mostrado na FIGURA 4.10. As dependências entre as tarefas podem ser preenchidas de forma manual nas estruturas *RT_Info*, ou podem ser obtidas automaticamente quando o usuário utiliza e registra seus objetos no Serviço de Eventos do TAO. Finalmente, a aplicação informa ao Serviço de Escalonamento que todas as tarefas a serem escalonadas já estão registradas.

Depois que o Serviço de Escalonamento possui o repositório de *RT_Infos* e as seqüências de chamadas de *RT_Tasks*, elas são analisadas para a derivação das *threads* de tempo real que compõe o sistema.

Nas suas primeiras versões o TAO suportava apenas escalonamento estático seguindo uma política *Rate Monotonic* de atribuição de prioridades. Entretanto, o escalonamento estático apresenta algumas desvantagens, como [SCH2000c]:

- Não suporta processamento distribuído de uma forma eficiente, muitos dos problemas de escalonamento em multi-processadores são muito complexos e de grande exigência computacional;
- Tarefas aperiódicas não são tratadas de uma forma eficiente;
- Visto que não há mudança de prioridades de uma forma fácil em tempo de execução, alocações de recursos devem ser feitas segundo as condições do pior caso.

Por estes motivos, entre outros, as novas versões do Serviço de Escalonamento do TAO já suportam escalonamento dinâmico (tal como EDF – *Earliest Deadline First*) assim como pode ser visto em [GIL2000].

4.6.6 Serviço de Eventos de Tempo Real

Na seção 3.4.2 foram mostradas as principais características e componentes do serviço de eventos padrão do CORBA, também denominado de COS (CORBA *Object Services*) *Event Service*. Ainda, o serviço de eventos traz vantagens em termos de flexibilidade em relação à forma de invocação padrão *(two-way*) de métodos do CORBA. Entretanto, no que se refere a tempo real, o serviço de eventos falta em algumas importantes características como: escalonamento e despacho de eventos de tempo real, processamento de eventos periódicos e correlações e filtragem centralizada. Por este motivo, o TAO acrescenta melhorias no serviço de eventos padrão no que se refere ao modelo *push*. Alguns dos itens contemplados pelo serviço de eventos de tempo real do TAO são [HAR97]:

- O serviço de eventos de tempo real permite que os consumidores e fornecedores de eventos especifiquem suas exigências e características de execução usando parâmetros de QoS (tais como tempo de execução no pior caso, período, etc), os quais são utilizados pelo mecanismo de despacho e pelas políticas de escalonamento para estabelecer ordem de despacho e estratégias de preempção.

- Foram acrescentados mecanismos de correlação e filtragem de eventos centralizados no canal de eventos de forma que os consumidores podem especificar relações lógicas de *OU* e *E* entre os eventos dependentes.
- Os consumidores podem especificar eventos dependentes de *timeouts*. Os pedidos de *timeout* são usados para propagar eventos de tempo em coordenação com o sistema de políticas de escalonamento. Além da forma normal do uso de *timeouts* (ou seja, receber eventos de *timeout* em certos intervalos), um consumidor pode requisitar o recebimento de um evento de *timeout* se as suas dependências não foram satisfeitas em um certo intervalo de tempo.

Arquitetura do serviço de eventos de tempo real

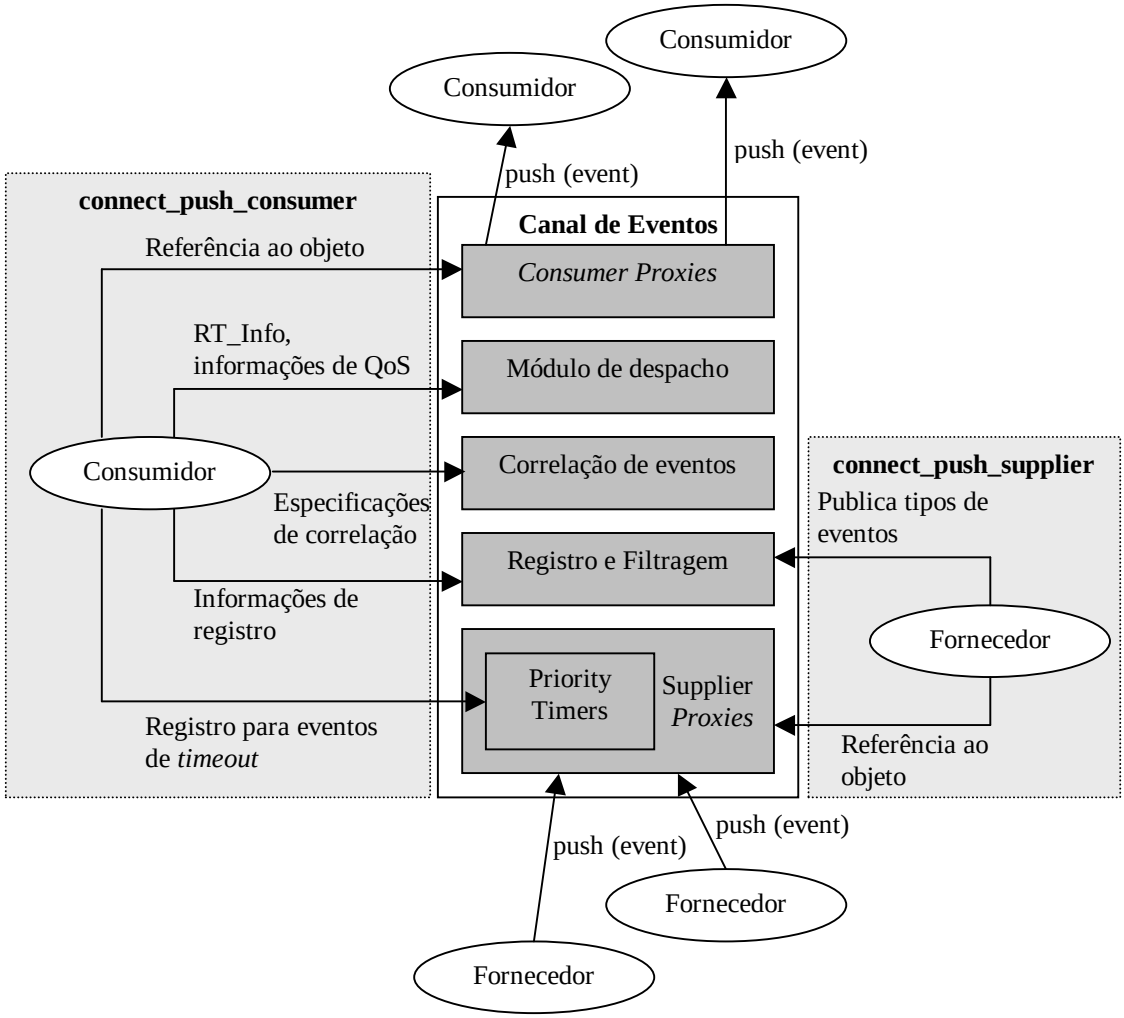


FIGURA 4.11 - Arquitetura do Serviço de Eventos de Tempo Real.

A FIGURA 4.11 mostra a arquitetura em mais alto nível da implementação do serviço de eventos de tempo real do TAO. A tarefa dos principais componentes desta arquitetura é apresentada a seguir [HAR97]:

- ***Event Channel*** (canal de eventos): de um modo geral ele atua como o canal de eventos da definição padrão do CORBA. Externamente provê duas interfaces,

ConsumerAdmin e *SupplierAdmin*, as quais permitem às aplicações obter objetos de administração para consumidores e fornecedores. São esses objetos de administração que permitem aos consumidores e fornecedores conectar e desconectar do canal. Internamente o canal é composto por diferentes módulos que encapsulam tarefas independentes do canal.

- **Módulo *Consumer Proxy:*** idêntica à interface *ConsumerAdmin* do COS *Event Service*. Provê métodos *factory* para criar objetos da interface *ProxyPushSupplier*, a qual é utilizada por consumidores para conectar e desconectar do canal. Esta interface foi estendida pelo TAO de modo a permitir que os consumidores registrem dependências de execução com o canal.
- **Módulo *Supplier Proxy***: idêntica à interface *SupplierAdmin* do COS *Event Service*. Provê métodos *factory* para criar objetos da interface *ProxyPushConsumer*, a qual é utilizada por fornecedores para conectar e desconectar do canal. Esta interface foi estendida pelo TAO de modo que os fornecedores pudessem especificar os tipos de eventos que eles geram. Estas informações são utilizados pelo módulo de filtragem e registro para criação de estruturas de dados para busca e registro de consumidores de forma eficiente e previsível. A interface *ProxyPushConsumer* também é utilizada para que os fornecedores enviem mensagens para os consumidores registrados através do método *push* desta interface.
- **Registro e filtragem**: o COS *Event Service* define o canal de eventos como um mecanismo que repassa todas as mensagens dos fornecedores para os consumidores. Há desvantagens quando, por exemplo, alguns consumidores estão interessados em apenas um certo subconjunto de eventos dos fornecedores. No serviço de eventos de tempo real do TAO é possível para os consumidores se registrar para um subconjunto particular de eventos. O canal utiliza este registro para filtrar os eventos dos fornecedores e repassar apenas aqueles eventos que interessam aos consumidores. Os consumidores podem definir interesse por eventos apenas de alguns fornecedores, eventos de um determinado tipo (foi necessária a criação de um esquema completo de tipo de eventos para o serviço de eventos do TAO), ou uma combinação destas duas formas. O módulo de registro e filtragem ainda oferece as operações de *resume* e *suspend* para os consumidores, o que lhes permite cancelar temporariamente o recebimento de eventos de uma forma menos custosa.
- ***Priority Timer Proxy***: faz parte do módulo *Supplier Proxy* e gerencia todos os *timers* registrados no canal. Quando um consumidor se registra para eventos de *timeout* o *priority timer proxy* coopera com o escalonador de *runtime* para assegurar que os *timeouts* são despachados de acordo com a prioridade do seu correspondente consumidor.
- **Correlação de eventos**: os consumidores podem exigir que certos eventos ocorram antes que ele possa continuar. Para implementar esta funcionalidade os consumidores podem especificar semânticas de conjunção (E – onde o canal de eventos informará o consumidor quando “todos” os eventos especificados nas dependências tiverem sido recebidos) ou de disjunção (OU – onde o canal de eventos informará o consumidor quando “qualquer” um dos eventos especificados nas dependências tiver sido recebido).
- **Despacho**: o módulo de despacho determina quando os eventos devem ser enviados aos consumidores e é responsável por realizar esse envio. Para garantir que os consumidores executem em tempo para respeitar seus

deadlines, este módulo interage com o serviço de escalonamento do TAO (o qual foi discutido na seção 4.6.5).

4.6.7 Object Adapter de Tempo Real

Os pontos chave no desenvolvimento de um *Object Adapter* para ORBs de tempo real são: (i) determinar como demultiplexar as requisições dos clientes de forma eficiente, escalável e previsível, e (ii) determinar como compartilhar a capacidade de processamento do sistema entre as diferentes operações das aplicações.

O *Object Adapter* do TAO pode ser configurado para implementar mecanismos específicos para despachar requisições dos clientes de acordo com políticas de escalonamento específicas de cada aplicação. De modo a compartilhar os recursos de processador entre as operações é utilizado o Serviço de Escalonamento de Tempo Real descrito na seção 4.6.5. Uma das estratégias do Object Adapter do TAO é uma variação do escalonamento periódico *rate monotonic* implementado com *threads* e *upcalls* de tempo real.

Requisição de Despacho e Escalonamento de Tempo Real

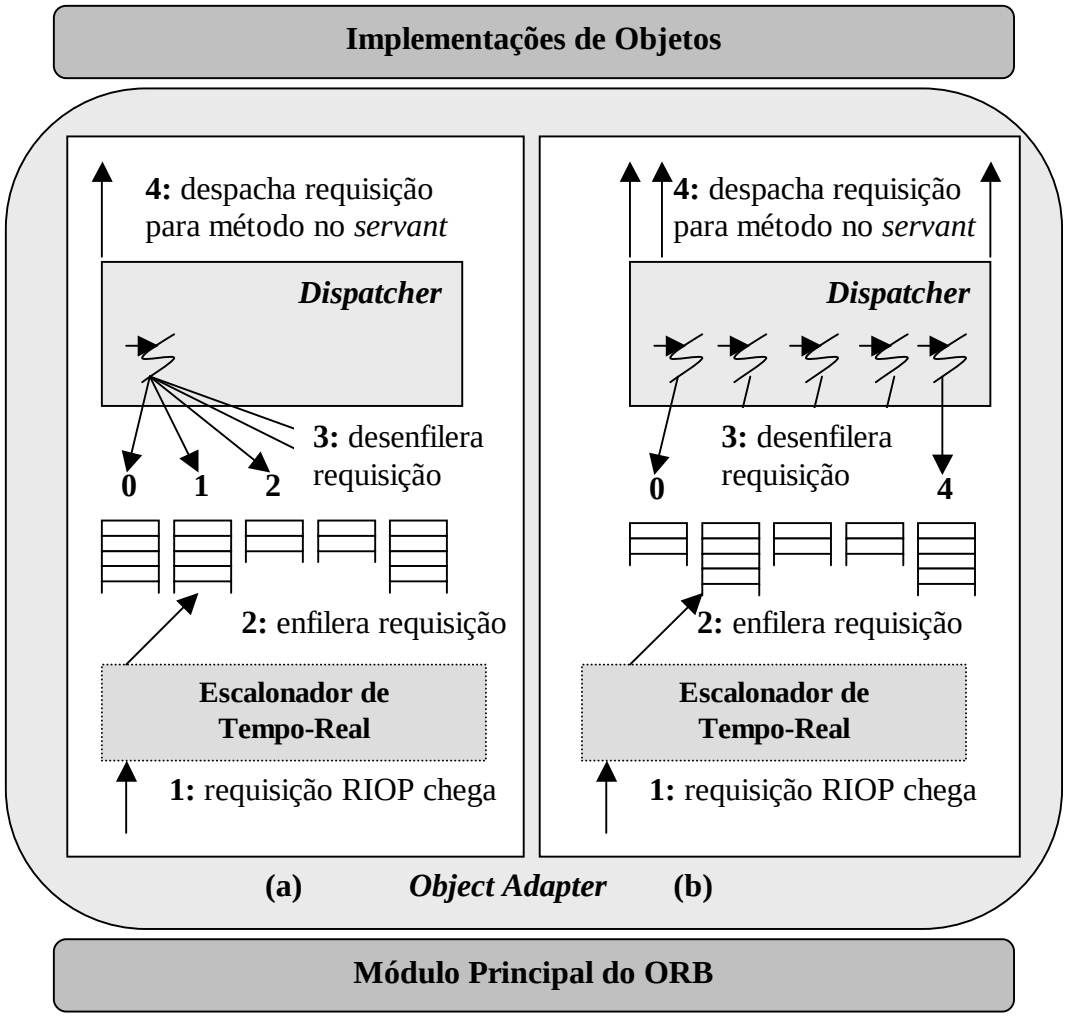


FIGURA 4.12 - Mecanismos de despacho do Object Adapter do TAO.

Conforme pode ser visto na FIGURA 4.12, o *Object Adapter* do TAO possui uma referência ao Escalonador de Tempo Real, o qual despacha as requisições dos clientes de acordo com a política de escalonamento de tempo real. O Escalonador de Tempo Real mapeia as requisições dos clientes para prioridades de *threads*. Na sua implementação inicial, a qual suporta aplicações de tempo real determinístico, é feita uma consulta a tabelas de prioridades de requisições geradas de forma *offline*.

A FIGURA 4.12 mostra os seguintes mecanismos de despacho suportados *pelo Object Adapter*:

- **Despacho RTU (com interrupção protelada)**: a FIGURA 4.12 (a) mostra uma implementação baseada em uma única *thread* responsável por enfileirar e despachar todas as requisições. Este mecanismo exige que os consumidores cooperativamente interrompam-se quando consumidores de mais alta prioridade se tornam aptos a rodar. Este modelo de interrupção protelada é baseado no mecanismo de concorrência RTU (*Real-Time Upcall*). Seus grandes benefícios residem principalmente na habilidade de reduzir as trocas de contexto, sincronização e movimentações de dados decorrentes da utilização de implementações baseadas em várias *threads*. Um dos problemas desse modelo é não ter uma interrupção (preempção) automática; ele exige cooperação das aplicações para realizar o despacho. Em alguns casos pode ocorrer uma inversão de prioridade até que seja feita checagem da mesma de modo a decidir pela preempção da atividade.
- **Despacho de *threads* de tempo real**: a FIGURA 4.12 (b) mostra uma implementação que aloca uma *thread* de tempo real (ou um estoque de *threads*) para cada fila de prioridades de requisições de clientes. Esta forma de implementação pode tirar vantagem do suporte à preempção do sistema operacional. Assim, quando uma *thread* de alta prioridade se torna pronta o próprio sistema operacional automaticamente interrompe *threads* de mais baixa prioridade para que aquelas possam ser executadas.

Otimização na demultiplexação

ORBs convencionais realizam demultiplexação de requisições dos clientes através de um mecanismo em camadas. A demultiplexação em níveis ou camadas pode trazer queda de desempenho devido ao aumento do número de tabelas internas que devem ser consultadas enquanto a requisição do cliente vai subindo pelas camadas de processamento que compõe o sistema do ORB. Além disso, podem ocorrer inversões de prioridade nesse processo devido ao fato de que informações de QoS importantes ainda estão inacessíveis para os níveis inferiores de serviços e protocolos.

Desta forma, para tentar aumentar a eficiência e o desempenho da implementação, o TAO apresenta dois mecanismos para processar as requisições dos clientes:

- ***Hash* perfeito**: a estratégia de *hash* perfeito mostrada na FIGURA 4.13 (a) é um mecanismo de demultiplexação em camadas com dois passos. Esta estratégia utiliza uma função de hash perfeito gerada automaticamente para localizar a implementação do objeto e uma segunda para localizar a operação. Em muitos sistemas de tempo real determinísticos, onde implementações de

objetos e operações podem ser configuradas estaticamente, pode ser utilizada esta técnica para a localização de objetos e operações.

- **Demultiplexação ativa:** nesta estratégia, mostrada na FIGURA 4.13 (b), o cliente passa um identificador que diretamente endereça a operação de um objeto. Este identificador é configurado no cliente quando o objeto referenciado é registrado no Serviço de Nomes ou no Serviço de Negociação (*Trading Service*). Assim que a requisição chega no servidor, o *Object Adapter* utiliza o identificador fornecido no cabeçalho da requisição CORBA para localizar o objeto e a operação em um único passo.

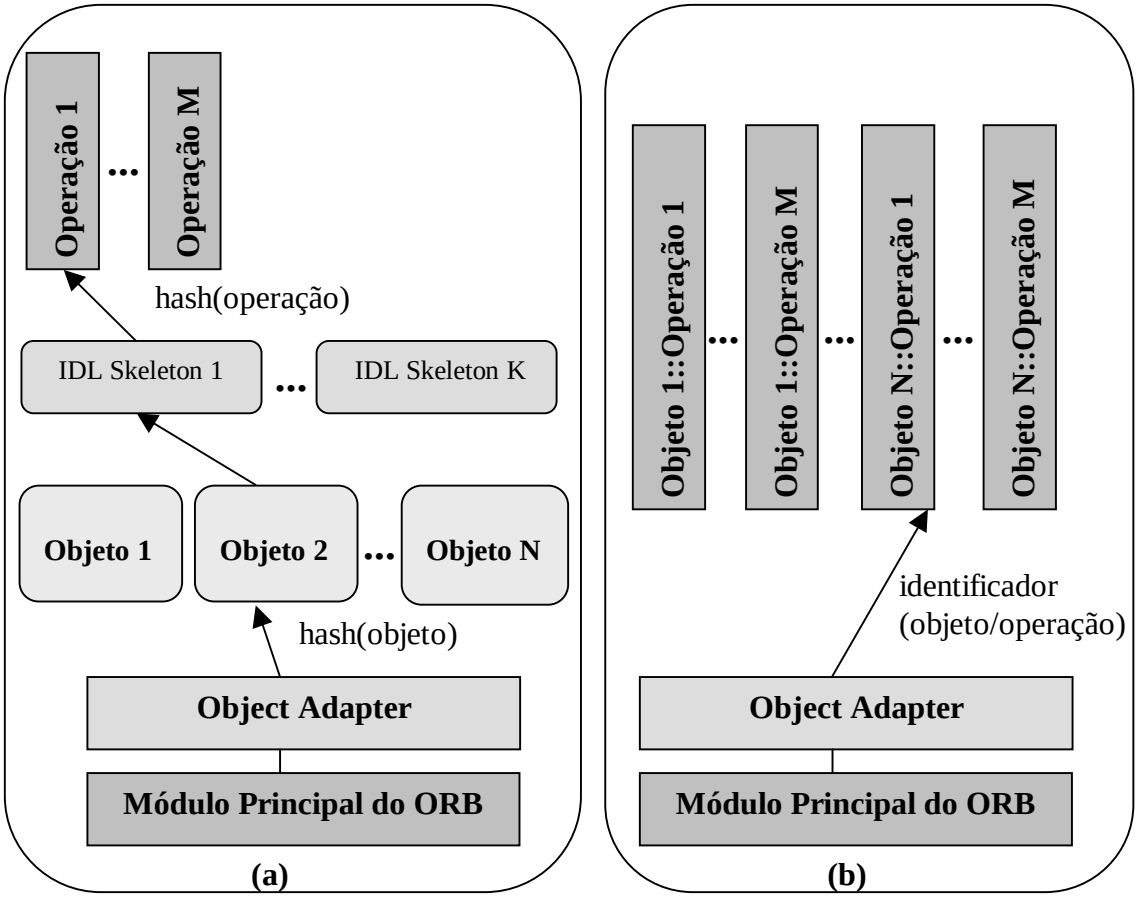


FIGURA 4.13 - Estratégias de demultiplexação otimizadas.

4.6.8 Compilador IDL e camada de apresentação

A camada de apresentação transforma os parâmetros das operações de uma representação de alto nível para uma representação de baixo nível (*marshaling*) e vice-versa (*demarshaling*). Visto que a camada de apresentação é um dos grandes gargalos do sistema de comunicação, minimizando operações custosas como alocação dinâmica de memória e grande cópias de dados, pode-se aumentar o desempenho do ORB.

No TAO, a camada de apresentação é formada pelos *stubs* do lado do cliente e pelos *skeletons* do lado do servidor, os quais são gerados automaticamente pelo

compilador IDL. Além de reduzir o potencial de inconsistências entre os *stubs* e *skeletons*, o compilador IDL, *Flick*[4], ainda possui as seguintes otimizações [SCH2000c]:

- **Uso reduzido de memória dinâmica**: são analisadas as exigências de armazenamento para todas as mensagens trocadas entre cliente e servidor. Assim, o espaço suficiente é alocado de antemão, evitando a necessidade de testes repetitivos para determinar se há espaço suficiente, e caso não haja, estender o espaço disponível dinamicamente.
- **Reduzida cópia de dados:** o compilador IDL analisa quando é possível fazer cópias em bloco para tipos de dados atômicos ao invés de copiá-los individualmente. Isto evita excessivas cópias de dados minimizando o número de instruções de *load* e *store*.
- **Redução na sobrecarga de chamadas de funções:** *stubs* pequenos são otimizados com a utilização de funções *inline*. Melhorando, assim, o desempenho de funções que fazem uso desses *stubs*.

O TAO estende o compilador IDL *Flick* para gerar e configurar múltiplas estratégias para empacotamento e desempacotamento de tipos IDL. Assim é possível gerar tanto *stubs* e *skeletons* interpretados como compilados. Esta flexibilidade permite obter um bom resultado entre o código interpretado (mais lento, mas mais compacto em tamanho) e o código compilado (mais rápido, mas maior em tamanho).

Da mesma forma, o TAO pode realizar armazenamento temporário de dados da aplicação que sejam utilizados repetidamente. Assim pode-se aumentar o desempenho quando dados como esses são transferidos em seqüências de requisições. Mas isto exige que o ORB faça uma análise de fluxo do código da aplicação para determinar que campos das requisições serão armazenados temporariamente.

Apesar do fato de que estas técnicas podem reduzir o tempo de empacotamento das requisições para o caso normal, aplicações com exigências de serviços de tempo real rígidas freqüentemente consideram apenas a execução de pior caso. Assim, a análise de fluxo deve ser feita apenas sob certas circunstâncias.

4.6.9 Otimizações no gerenciamento de memória

Sistemas CORBA convencionais sofrem pelo excessivo uso de gerenciamento de memória dinâmica e sobrecarga nas cópias de dados. Por exemplo, o sistema de E/S e o módulo principal do ORB alocam uma área de memória para cada requisição de cliente que chega e o sistema de E/S normalmente copia o conteúdo das áreas por ele alocadas para as áreas alocadas pelo ORB. Procedimentos de decodificação de implementações do GIOP/IIOP padrões alocam memória para armazenar os parâmetros decodificados das requisições. Finalmente, *skeletons* IDL alocam dinamicamente e removem cópias dos parâmetros das requisições antes e depois de chamadas a métodos das implementações de objetos, respectivamente.

[4] Flick é um novo compilador IDL em desenvolvimento no Departamento de Ciência da Computação da Universidade de Utah, EUA. Explora conceitos dos compiladores de linguagens de programação tradicionais para criar um compilador IDL mais flexível e otimizado. É dividido em três fases (*front end, presentation generator, back end)* implementadas como bibliotecas dinâmicas em C/C++ provendo abstrações para construções IDL, tipos de dados alvo e tipos de mensagens. As bibliotecas implementam funções genéricas que servem como base para especializações necessárias para comportamentos particulares em termos de IDL, mapeamento de linguagem, formato de mensagens e meio de transporte [EID97].

Estas políticas de gerência de memória são importantes em algumas circunstâncias. Entretanto, excessivas cópias desnecessárias aumentam a sobrecarga da memória e barramento em aplicações que consomem seus parâmetros de entrada imediatamente sem altera-los. Em geral, a gerência de memória dinâmica é problemática para sistemas de tempo real determinísticos porque a fragmentação da memória alocada dinamicamente pode criar comportamento não–uniforme para mensagens de diferentes tamanhos e diferentes cargas de trabalho. Da mesma forma, excessivas cópias de dados podem reduzir drasticamente o desempenho fim-a-fim de um ORB *endsystem*[5].

O TAO foi projetado para minimizar e procurar eliminar cópias de dados entre as camadas do sistema ORB. Por exemplo, o sistema de gerenciamento das áreas de armazenamento (*buffers*) do TAO utilizam o adaptador de rede APIC (descrito brevemente na seção 4.6.2) para melhorar os sistemas operacionais convencionais com um sistema de gerenciamento de *buffers* com "zero cópias" de dados. No nível de *driver* o APIC interage diretamente com o barramento e outros *drivers* de E/S para transferir as requisições dos clientes da rede sem incorrer em cópias de dados adicionais.

Os conjuntos de áreas de armazenamento do APIC para sistemas de E/S suportam decodificação direta de requisições de clientes periódicas e aperiódicas em áreas de memória compartilhadas entre *threads* residentes de usuário e *kernel*. Estas APIs permitem às requisições dos clientes serem recebidas e enviadas da e para a rede sem incorrer em cópias desnecessárias de dados. Além disso, estes *buffers* podem ser alocados previamente e passados entre vários estágios do processamento do ORB, evitando a necessidade de uso de gerenciamento de memória dinâmica.

4.6.10 Especificação de QoS

De modo a suportar garantias de escalonamento e desempenho fim-a-fim, ORBs de tempo real devem permitir às aplicações especificar suas exigências de qualidade de serviço afim de que o sistema ORB possa garantir a disponibilidade dos recursos. Normalmente, em sistemas de tempo real determinísticos não distribuídos, a capacidade de processamento é um recurso escasso. Desta forma, o tempo necessário de processamento exigido pela requisição do processo cliente deve ser determinado antecipadamente de forma que o processador possa ser alocado de acordo. Para alcançar esse propósito as aplicações devem informar suas necessidades de tempo de processamento para o Serviço de Escalonamento *Offline* do TAO.

Em muitos casos, recursos diferentes da capacidade de processamento dizem respeito ao escalonamento em tempo de execução, mas nesta seção serão abordadas fundamentalmente as formas de especificação de exigências de processamento pelo processador. Serão apresentadas as interfaces IDL especificadas pelo *Real Time* IDL (RIDL) do TAO: a interface ***RT_Operation*** e a estrutura ***RT_Info***.

[5] Para o desenvolvimento do TAO, além da criação dos componentes necessários para a criação de um ORB de tempo real, tais como RT-POA e RT-ORB, ainda foram acrescentados módulos não só de *software*, mas também de *hardware* para aperfeiçoar o seu suporte a aplicações de tempo real. Exemplo desses módulos são um sistema de *buffers* voltado para a minimização de cópias de memória, um sistema de E/S e um placa para redes ATM de alta velocidade. Por este motivo, em muitos pontos de sua documentação, o TAO é considerado não apenas como um ORB, mas como um *endsystem*.

Interface *RT_Operation*

A interface ***RT_Operation*** é o mecanismo para transportar as exigências de processamento das tarefas executadas pelas operações das aplicações para o Serviço de Escalonamento do TAO. Na TABELA 4.2 segue a definição IDL dessa interface. Como pode ser visto, a interface ***RT_Operation*** contém a estrutura ***RT_Info***, a qual será descrita em seguida.

TABELA 4.2 - Interface RT_Operation.

interface **RT_Operation**{	
typedef Time::TimeT Time_t; typedef Time::TimeT Period_t;	

enum **Importance**{
INTERRUPT,
IO_SERVICE,
CRITICAL,
HARD_DEADLINE,
BACKGROUND
};
	Define a importância da operação, que pode ser utilizada como critério de desempate pelo escalonador quando outros parâmetros são iguais. INTERRUPT é de mais alta importância. BACKGROUND, de mais baixa importância não tem seu deadline garantido pelo Serviço de Escalonamento.

struct **Dependency_Info**{
any dependency_;
unsigned long number_of_calls_;
};
	dependency_ é na verdade uma estrutura RT_Info, mas o CORBA IDL não oferece uma forma de expressar dependência recursiva de dados.

struct **RT_Info**{
string operation_name_;
Time_t worst_case_execution_time_;
Time_t typical_execution_time_;
Time_t cached_execution_time_;
Period_t period_t;
Importance importance_t;
Time_t quantum_;
unsigned short number_of_threads_;
sequence <Dependency_Info>
operation_dependencies_;
};
	As informações da estrutura RT_Info definem os parâmetros de QoS de cada operação de uma aplicação.

};	

Estrutura *RT_Info*

Aplicações que utilizam o TAO devem especificar todas as suas exigências de recursos escalonáveis. Estas informações de QoS são passadas para o TAO antes da execução do programa. No caso dos recursos de processamento, eles são expressos através da estrutura ***RT_Info.*** A TABELA 4.3 mostra com mais detalhes os atributos desta estrutura.

TABELA 4.3 - Atributos da da estrutura RT_Info.

operation_name_	Nome que identifica unicamente a operação.
worst_execution_time_	É o tempo máximo que uma RT_Operation exige. É utilizado em análise de escalonamento conservativa para aplicações com exigências rígidas de tempo.
typical_execution_time_	É o tempo normal de execução da operação. Pode ser útil em análise de escalonamento de sistemas de tempo real estatísticos. Ainda não é utilizado palo Serviço de Escalonamento determinístico do TAO.
cached_execution_time_	Caso uma operação possa fornecer um resultado que pode ser armazenado temporariamente em resposta a requisição, então o valor do tempo de execução, quando em *cache*, é determinado com um valor diferente de zero. Durante a execução, o custo do tempo máximo de execução é somado apenas uma vez caso o armazenamento temporário esteja habilitado. Nos outros casos o tempo será dado pelo tempo de execução em *cache*.
period_t	É o tempo mínimo entre sucessivas iterações da operação. Caso a operação execute em um objeto ativo, com múltiplas *threads* de controle, pelo menos uma delas deve executar a operação respeitando o período determinado. Caso o período esteja definido como zero a operação é totalmente reativa, ou seja, ela é apenas chamada como resposta para algum cliente.
importance_t	A importância é utilizada pelo Serviço de Escalonamento como um critério de desempate.
quantum_	Caso uma operação tenha sua importância definida como BACKGROUND, ela poderá ser retirada do processador a qualquer momento pelo *Object Adapter*. O valor que pode ser especificado para o atributo *quantum_* define o tempo máximo que uma tarefa que roda em BACKGROUND pode permanecer no processador. Quando este período termina, ela é automaticamente retirada, para que outra possa ser executada, fornecendo um escalonamento por fatia de tempo em um esquema de melhor esforço.
operation_dependencies_	É um vetor de estruturas *RT_Info*, uma para cada operação que é chamada diretamente por esta *RT_Operation*. Este vetor é utilizado pela análise de escalonamento para identificar as *threads* do sistema: cada operação separada que é chamada no vetor indica uma *thread*. Ainda é utilizado para cálculo do tempo de execução.

Os atributos descritos acima são utilizados para análise *Rate Monotonic*. As estruturas do RIDL especificam as características de tempo de execução das operações

dos objetos, e estas informações são utilizadas pelo TAO para (i) validar a viabilidade[6] do escalonamento e (ii) alocar recursos do sistema ORB e de rede.

4.7 Considerações Finais

Muitos sistemas de tempo real são inerentemente distribuídos, e várias soluções desde sistemas operacionais a linguagens de programação de tempo real são utilizadas para seu desenvolvimento. A possibilidade da utilização de um sistema aberto, que promete ser independente de plataforma e linguagem e onde há uma maior facilidade de reuso, é muito atraente. Entretanto, vários problemas ligados à ausência de meios de definir e garantir os requisitos de tempo e QoS e imprevisibilidade do *middleware* impediam a direta utilização do CORBA.

A especificação do RT-CORBA procura solucionar várias dessas deficiências através da introdução de um conjunto de interfaces para *Threads*, *ThreadPools*, elementos de sincronização e meios de especificar prioridade e protocolos de transporte preferidos. Adicionalmente, foi definido um serviço de escalonamento baseado em prioridades fixas e, atualmente, encontra-se em fase final de especificação uma versão de serviço de escalonamento dinâmico.

Por outro lado, o RT-CORBA pode ser considerado ainda como uma especificação em desenvolvimento. Algumas de suas melhorias em relação ao CORBA ainda estão sendo trabalhadas pelas comissões responsáveis da OMG. A ausência de algumas determinações mais claras e específicas causa a disseminação de implementações no mercado incompatíveis entre si, e muitas vezes distantes da própria especificação do RT-CORBA. Tentando contornar definições incompletas, ORBs como o TAO [TAO2000] e o Vertel [VER2000], acabam por criar soluções próprias. Apesar disso, a inclusão do RT-CORBA nas especificações do CORBA 3.0 é uma promessa de que seu desenvolvimento continuará e o número de implementações condizente com seus detalhes crescerá a ponto de se tornar um verdadeiro padrão para o desenvolvimento de aplicações distribuídas de tempo real.

Uma das implementações mais completas do RT-CORBA é o TAO, *The* ACE ORB. Ele foi projetado para ser um sistema final CORBA para aplicações que necessitem de garantias na qualidade de serviço e de tempo. Dentre os elementos de sua arquitetura foram destacados aqui: subsistema de E/S, protocolo GIOP de tempo real, Serviço de Escalonamento e Serviço de Eventos, *Object Adapter* de tempo real, otimizações de memória, compilador IDL e forma de representação de QoS, além do próprio módulo principal do ORB.

Apesar de todas as otimizações e melhorias que objetivam a diminuição de problemas como imprevisibilidade e inversão de prioridades, o TAO sendo um *middleware* não consegue contornar as deficiências de um sistema operacional convencional no que se refere a garantias de tempo e serviço. É deixado claro em vários pontos da documentação que um sistema de tempo real é requerido para que a qualidade de serviço e os requisitos temporais exigidos pelas aplicações e especificados para o TAO possam ser supridos por este.

---

[6] O termo viabilidade de um escalonamento, neste caso, é utilizado para especificar um escalonamento no qual todos os *deadlines* das tarefas escalonadas são respeitados.

5 Mecanismo de Adaptação Proposto

Apresentadas suas principais características no capítulo 4, o RT-CORBA (um padrão da OMG) e o TAO (através de características próprias que em alguns momentos fogem do padrão) procuram contornar os problemas que o CORBA possui quando é utilizado no desenvolvimento de aplicações de tempo real. As funcionalidades mostradas procuram disponibilizar e adequar esse *middleware* também para aplicações de *hard real time*. O sucesso das soluções apresentadas não depende apenas do próprio *middleware*, mas também de todo o suporte de sistema operacional e rede no qual a aplicação de tempo real irá ser executada.

Quando não se dispõem de um suporte adequado para a execução de aplicações de tempo real, como um sistema operacional de propósito geral, ou uma rede como a Internet, a correta execução de aplicações de *hard real time* fica mais difícil. Entretanto, é possível aumentar a abrangência de condições em que as aplicações de *soft real time* podem ser executadas, principalmente com a utilização de técnicas adaptativas. Algumas dessas técnicas foram apresentadas no capítulo 2 e variações dessas abordagens também podem ser aplicadas no ambiente RT-CORBA.

Este capítulo tem por objetivo apresentar uma proposta de mecanismo de adaptação para o ambiente RT-CORBA. O propósito deste mecanismo é permitir um controle da degradação do sistema de tempo real em momentos de sobrecarga. Inicialmente é apresentada uma descrição dos princípios básicos da proposta, seguindo com os detalhes de sua implementação (protótipo) utilizando o ORB TAO.

5.1 Descrição

Considerando que muitas aplicações de tempo real têm sua estrutura baseada em atividades periódicas, optou-se por buscar entre as propostas da literatura aquelas mais direcionadas a esse tipo de atividade de forma a servir de base para o mecanismo apresentado aqui. Assim como em Goel et al. [GOE99] utilizou-se realimentação para controlar o sistema em caso de sobrecarga, de forma que o próprio sistema fornece os dados necessários para que o controle da adaptação possa atuar no mesmo. O mecanismo básico do conceito de realimentação pode ser visto na FIGURA 5.1.

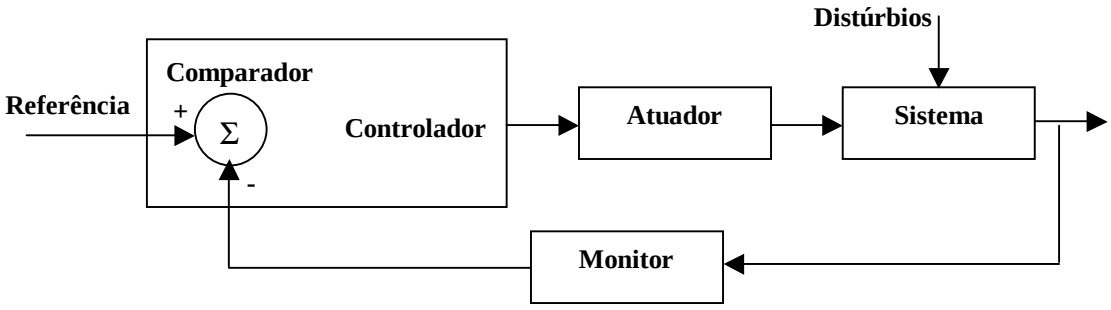


FIGURA 5.1 - Mecanismo básico de realimentação.

A coleta de informações do sistema pode ser feita de várias maneiras. Em Nett et al.[NET98] sensores (invocações de métodos que informam o quanto o processamento da tarefa progrediu) são inseridos no código da aplicação, ou do *middleware* ou do sistema operacional (de forma automática ou não) de modo a obter as informações necessárias sobre o sistema. Esta abordagem pode incorrer em um excesso de informações a serem processadas pelo controle da adaptação, assim como foi detectado pelos autores daquele artigo.

Dessa forma, na proposta aqui apresentada procurou-se efetuar a coleta de informações bem como implementar um mecanismo de adaptação que afetasse o mínimo possível o próprio processamento da aplicação. Assim como sugere Mcelhone & Burns [MCE00] o custo computacional de executar um algoritmo de adaptação no mesmo processador que as tarefas obrigatórias ou opcionais deve ser levado em consideração, de modo a procurar alternativas que sobrecarreguem menos o sistema.

Além dos demais trabalhos que utilizam realimentação, a proposta deste trabalho foi inspirada em Lu et al. [LU2000]. Da mesma forma que naquele trabalho, a atuação no sistema é dependente das informações que são coletadas junto às atividades que compõem o sistema e analisadas por um processo periódico, que neste trabalho será denominado de *AdaptiveService*. Lu et al. [LU2000] analisa, para adaptação do sistema, as informações de taxa de utilização do sistema (U(t)) e a taxa de perda de *deadline* (MR(t) – *miss ratio function*) das tarefas em uma janela de medição (MW – *miss-ratio window*). A medida da taxa de perda de deadline é bastante adequada para sistemas com *deadline* firme (*firm-deadline*), ou seja, onde não existe benefício em executar uma tarefa após o seu *deadline*.

Ao contrário daquela abordagem, que foi criada para um sistema monoprocessado, estamos tratando de sistemas distribuídos e ainda buscamos neste trabalho considerar tarefas não tão exigentes, onde mesmo perdendo seus prazos a execução das tarefas ainda é importante. Assim, a forma de quantificar o desempenho do sistema foi através da medida comparativa entre os *deadlines* das tarefas com os tempos de resposta efetivamente observados. Esta medida comparativa foi denominada de *delay profile* (DY(t)), o qual é observado após uma janela de medição denominada de *delay window* (DW). Neste trabalho, o *deadline* de cada tarefa está sendo considerado como o período da mesma (D=P), portanto, DY(t) pode ser expresso por:

$$DY(t) = \sum_{t-DW}^{t} \frac{(R_i - D_i)}{N}$$

t: instante de tempo atual
DW: *delay window*, janela de tempo para cálculo de DY(t)
R_i: tempo de resposta da tarefa *i* para uma determinada ativação
D_i: *deadline* da tarefa
N: número de conclusões (de todas as tarefas) na janela DW

A atuação é feita sobre os períodos das tarefas que compõe o sistema. Esta atuação poderia ser feita de forma específica para cada tarefa. Possuindo um fator de adaptação calculado especialmente para cada tarefa diferenciando-as conforme sua prioridade, por exemplo. Considerando que todas as tarefas foram escalonadas (utilizando uma política como *Rate Monotonic*, por exemplo) o fator de diferenciação entre as tarefas já foi atribuído como sendo sua prioridade, caso a plataforma (sistema operacional e/ou *middleware*) respeite esta atribuição. Uma atuação especifica a cada

tarefa não seria necessária. Assim, DY(t) é utilizado para o cálculo de um "*fator*" de atuação sobre o período de cada tarefa da seguinte forma:

$$fator = K_P * E(t)$$
$$fator = K_P * (DY(t) - SDY)$$
$$fator = K_P DY(t) - K_P SDY$$

Considera-se que E(t) é o valor do erro do sistema em um determinado instante, o qual pode ser expresso por DY(t)-SDY, onde SDY é o atraso médio desejável para o sistema. K_P (constante proporcional) e SDY são valores atribuídos pelo projetista e/ou programador do sistema segundo cálculos matemáticos ou experimentação. Quanto menor o valor de K_P mais suave será a atuação no sistema.

O *fator* é então utilizando pelo *AdaptiveService* pelo cálculo do período efetivo de cada tarefa conforme:

$$P_{efetivo} = P_{mínimo} + fator(P_{máximo} - P_{mínimo})$$

Além dos valores de K_P, SDY e DW, a forma de o projetista/programador especificar a atuação se estende para os valores de $P_{mínimo}$ (período mínimo) e $P_{máximo}$ (período máximo) de cada tarefa. Segundo a fórmula acima, o *AdaptiveService* é capaz de variar o período de cada tarefa (período efetivo) segundo o *fator* calculado anteriormente sempre dentro dos limites estabelecidos para o período mínimo e máximo de cada uma.

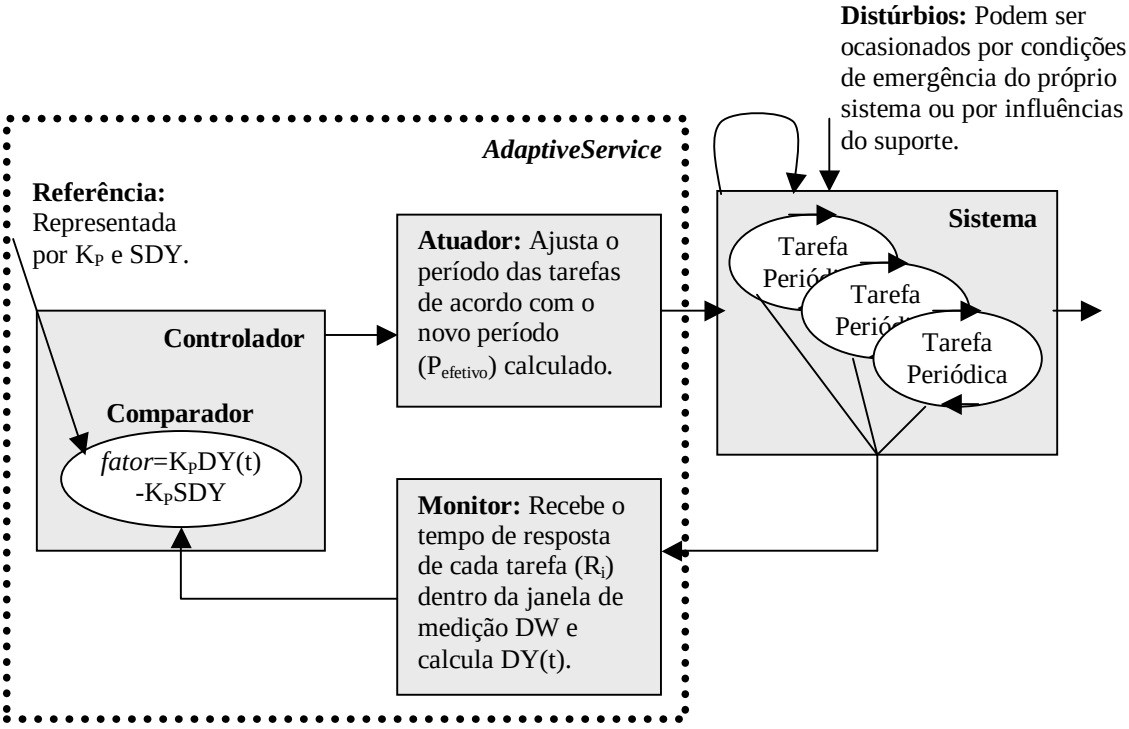


FIGURA 5.2 - Mecanismo de adaptação proposto.

Por conseguinte, o projetista é capaz de configurar a atuação de modo que ela seja mais suave ou mais forte, conforme resultado de cálculos ou experiências. Ainda,

ajustando-se o período mínimo e máximo iguais para uma tarefa é possível especificar que esta tarefa não pode estar sujeita a adaptação, por ser uma tarefa crítica, por exemplo. Isto permite que o mecanismo proposto seja utilizado também em situações onde existem tarefas de *hard real time* e tarefas de *soft real time*.

O mecanismo de adaptação proposto neste trabalho pode ser representado, utilizando como base a FIGURA 5.1 que mostra o mecanismo básico de realimentação, através da FIGURA 5.2. Nesta figura pode-se ver que o *AdaptiveService* engloba as atividades de monitoração, atuação e controle. Para efeitos de adaptação considera-se que o sistema seja formado apenas por tarefas periódicas. Tarefas não periódicas podem existir, mas não há necessidade de envio de informações sobre essas tarefas para o serviço de adaptação visto que esse não irá atuar sobre elas.

5.2 Implementação do mecanismo de adaptação no contexto do ORB TAO

O mecanismo de adaptação proposto se destina a aplicações compostas por tarefas periódicas. No entanto, aplicações reais também podem conter tarefas não periódicas e passivas. Assim, o diagrama de classes da implementação do mecanismo de adaptação, que pode ser visto na FIGURA 5.3, contempla também este tipo de tarefa.

A hierarquia de classes (interfaces) mostrada na FIGURA 5.3 (aquelas classes que apresentam a relação de herança entre si) é definida em IDL. A relação de herança entre interfaces em IDL apenas garante que a interface derivada será compatível em termos de definição de atributos e métodos com relação à interface base. A implementação dos métodos pode ser completamente diferente, ou seja, o comportamento de métodos com a mesma assinatura em interfaces que apresentam a relação de herança pode ser diferente [HEN99].

Isto significa que no código gerado pelo compilador IDL (no caso deste trabalho, código C++) não será indicado que a implementação da interface ***PeriodicTask*** é derivada da implementação da interface ***Task***, por exemplo. Porém, com o intuito de manter a compatibilidade também de implementação entre essas interfaces, o código C++ gerado para cada interface foi alterado de forma que as classes mantivessem sua relação de herança também em implementação. Os detalhes da implementação dos principais elementos da FIGURA 5.3 são detalhados nos próximos itens desta seção.

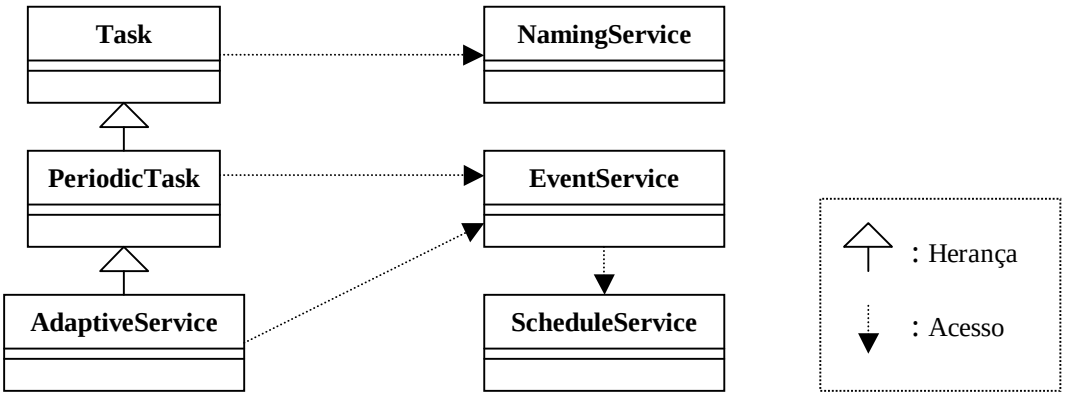


FIGURA 5.3 - Diagrama da implementação do modelo de adaptação.

5.2.1 Task

A interface ***Task*** tem por objetivo representar tarefas de tempo real não periódicas ou passivas. Qualquer tarefa não periódica deve ser definida como uma interface derivada desta. A TABELA 5.1 mostra a definição IDL desta interface.

A implementação da classe *Task* foi feita de forma que seja uma classe abstrata, ou seja, não pode ser instanciada diretamente. É, portanto, necessária a criação de uma classe derivada. Isto se deve ao fato de que cada tarefa pode possuir métodos diferentes que podem ser invocados. Dessa forma, a classe *Task* apenas fornece um esqueleto a ser utilizado pelas classes derivadas.

TABELA 5.1 - Representação IDL da interface Task.

interface Task{	
string get_name();	Método que retorna o nome (o qual é registrado no serviço de nomes) do objeto.
boolean activate();	Responsável por ativar o objeto no POA e registrar seu nome no serviço de nomes.
boolean deactivate();	Desativa o objeto do POA e retira seu nome do serviço de nomes.
};	

A interface *Task* é implementada como um *servant* que irá encarnar um objeto CORBA. No momento de sua ativação registra um nome no serviço de nomes (***NamingService***) para que os objetos ativos do sistema possam encontrá-los e realizar invocações de métodos.

Cada tarefa, objeto CORBA, é ativada em um POA próprio criado como um filho do *root_poa* sendo configurada a política *PriorityModelPolicy* para o valor *ClientPropagated*, de forma a respeitar a prioridade dos objetos que invocam seus métodos. Logo após sua criação é feita a ativação do objeto (invocação ao método *activate*) que fará com que o *servant* encarne o objeto CORBA.

5.2.2 PeriodicTask

A interface *PeriodicTask* representa as tarefas periódicas que podem compor um sistema de tempo real. Assim como *Task*, essa classe foi implementada como uma classe abstrata, que não pode ser diretamente instanciada. Uma tarefa periódica (*PeriodicTask*) é, em última instância, um consumidor que se registra no serviço de eventos (*EventService)* do TAO para receber eventos de *timeout* em intervalos específicos de tempo. Este intervalo é o período definido na *RT_Info* da tarefa na fase *offline* do escalonamento.

A TABELA 5.2 apresenta a definição desta interface em IDL.

TABELA 5.2 - Representação IDL da interface PeriodicTask.

interface PeriodicTask : Task,RtecEventComm::PushConsumer{	É derivada da interface *Task* e também da interface *PushConsumer*.
unsigned long get_period();	Retorna o período originalmente configurado para a tarefa.
unsigned long get_handle();	Retorna o *handle* (identificador) da *RT_Info* da tarefa.
boolean reconfig(in long period);	Altera o período da tarefa, ou seja, faz uma reconexão com o serviço de eventos de forma a atualizar o período do recebimento de eventos de *timeout*.
boolean prepare();	Método que deve ser sobrescrito pelas classes derivadas. Deve efetuar procedimentos de preparação para a execução da tarefa, como obter referências a objetos remotos através do serviço de nomes, por exemplo.
boolean body();	Este método também deve ser sobrescrito pelas classes derivadas e é executado a cada ciclo da tarefa.
};	

Por ser derivada da interface *Task* as tarefas periódicas também possuem métodos de ativação e desativação, porém, estes métodos não possuem a mesma implementação que naquela classe. Isto se deve ao fato de que as políticas configuradas para o POA das tarefas periódicas são diferente. Além disso, *PeriodicTask* é um *PushConsumer* e portanto possui o método *push* definido. A implementação deste método fica a carga da própria classe *PeriodicTask* e não de uma classe derivada, porque é ele que possui os controles de tempo de resposta da tarefa que serão informados para o serviço de adaptação.

Ainda, cabe ressaltar que cada *PeriodicTask* é ativado em um POA próprio (filho de *root_poa*) que tem a política *PriorityModelPolicy* configurada como *ServerDeclared* (de forma a utilizar a prioridade definida em sua *RT_info*) e um *ThreadPool* com número de *threads* estáticas e dinâmicas que pode ser alterado pelo programador no momento de criação de instâncias.

Além da prioridade, as outras informações necessárias para a configuração de cada tarefa periódica, como seu período, pior tempo de execução, etc. devem ser cadastradas na fase *offline* do escalonamento como *RT_Infos*. As tarefas não periódicas (*Task*) também devem ser cadastradas em *RT_Infos* de modo que também se possa informar o escalonador sobre as dependências das tarefas periódicas em relação às tarefas não periódicas. O escalonador atribuirá prioridades também para as tarefas não periódicas. Entretanto, com o objetivo de minimizar as inversões de prioridades entre as tarefas periódicas, as tarefas não periódicas, como citado anteriormente, foram configuradas possuindo a política de *PriorityModelPolicy* com o valor de *ClientPropagated*, ou seja, as tarefas não periódicas serão executadas com a prioridade da tarefa periódica que a invocou.

5.2.3 AdaptiveService

O serviço de adaptação (*AdaptiveService*) foi definido como uma interface derivada de *PeriodicTask*, ou seja, também foi implementado como uma tarefa periódica que precisa ser cadastrada como uma *RT_Info* na fase *offline* do escalonamento. Seu período, o qual assume-se igual a DW, pode ser configurado na *RT_Info*.

A TABELA 5.3 apresenta a definição da interface do serviço de adaptação em IDL.

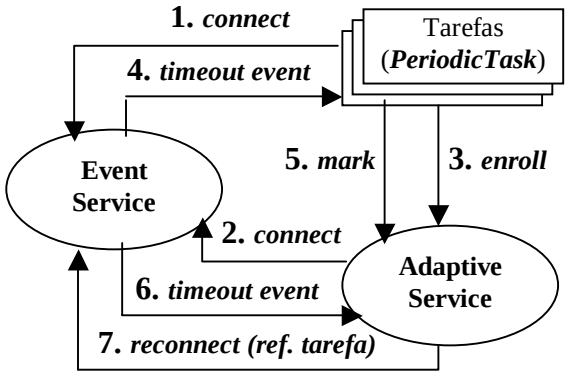
TABELA 5.3 - Representação IDL da interface AdaptiveService.

interface AdaptiveService : PeriodicTask{	O serviço de adaptação é uma tarefa periódica e por este motivo é derivado da interface *PeriodicTask*.
long enroll(in RtecEventComm::PushConsumer consumer_ref, in RtecEventChannelAdmin::ProxyPushSupplier proxy_supplier, in unsigned long handle, in unsigned long normalperiod, in unsigned long minperiod, in unsigned long maxperiod);	Método a ser invocado por qualquer tarefa periódica que deva ser considerada pelo mecanismo de adaptação. A referência para o *PushConsumer* e para o *ProxyPushSupplier* é necessária para que o próprio serviço de adaptação solicite a reconexão da tarefa junto ao serviço de eventos.
oneway void leave(in long enrollnumber);	Este método deve ser invocado pelas tarefas para solicitar a desconexão do serviço de adaptação.
oneway void mark(in long enrollnumber, in unsigned long exectime);	Método invocado por todas as tarefas periódicas que se registraram no serviço de adaptação para informar o tempo de resposta da última execução das mesmas.
};	

Visto que *AdaptiveService* é derivada de *PeriodicTask*, ela também herda os métodos daquela classe. Sendo que o método *prepare* não possui nenhuma funcionalidade para o serviço de adaptação, porém o método *body* contém a implementação do algoritmo de controle descrito anteriormente. Este método é invocado a cada evento de *timeout* enviado pelo serviço de eventos.

O serviço de adaptação é o responsável pela coleta dos dados junto às tarefas periódicas (as tarefas não periódicas não são contabilizadas) e posterior atuação no sistema. Este processo pode ser visto na FIGURA 5.4.

Para utilizar o serviço de escalonamento do TAO é necessário que a aplicação seja dividida em duas fases: *offline* e *runtime*. Na fase *offline* é feito o cadastramento das características temporais de cada uma das tarefas (e no caso do serviço de adaptação também) em estruturas chamadas de *RT_Infos*.



1. Cada tarefa se registra no serviço de eventos para receber eventos de *timeout* em intervalos definidos pelo período cadastrado na *RT_Info* na fase *offline* da aplicação.
2. O serviço de adaptação também ser cadastra para eventos de *timeout* informando o período no serviço de eventos.
3. Cada tarefa cadastra suas informações de períodos máximo e mínimo (enviando também sua referência) no serviço de adaptação.
4. Depois das tarefas se conectarem, o serviço de eventos passa a gerar eventos de *timeout* em intervalos definidos pelo período da tarefa cadastrado na sua *RT_Info*.
5. Após cada ativação de uma tarefa, é medido e enviado ao serviço de adaptação o tempo de resposta para aquela ativação.
6. Em cada ativação, o serviço de adaptação calcula o valor de DY(t) para a janela DW e o novo valor do fator.
7. Caso o fator da ativação atual seja diferente da ativação anterior o *AdptiveService* solicita ao serviço de eventos a reconexão de cada uma das tarefas informando o novo período efetivo.

FIGURA 5.4 - Coleta de informações e atuação do Serviço de Adaptação.

No início da fase de *runtime* cada tarefa periódica deve obter uma referência para o serviço de adaptação e invocar o método *enroll* para se cadastrar junto ao mesmo, informando os valores de período mínimo e máximo admitido. Ainda, cada tarefa periódica deve ser implementada como uma instância de uma classe derivada de *PeriodicTask*. Isto porque esta classe provê mecanismos que obtêm o tempo de resposta da tarefa através de *timers* e automaticamente invocam o método *mark* do serviço de adaptação para informar o valor medido.

O serviço de adaptação tem, portanto, a função de, a cada ativação (em intervalos de DW), calcular o valor de DY(t) e o valor do *fator* de adaptação dos períodos das tarefas. O $P_{efetivo}$ calculado para cada tarefa é então utilizado para solicitar, junto ao serviço de eventos, a reconexão de cada tarefa para eventos de *timeout*, mas com este novo valor como intervalo entre ativações. O tempo de resposta do serviço de adaptação a aumentos de carga é, portanto, dependente do tempo de resposta do serviço de eventos para re-conectar os consumidores para novos intervalos de eventos de *timeout*. Situações que exijam respostas imediatas a aumentos repentinos de carga necessitariam de um serviço de eventos com maior rapidez no que se refere a reconexões de consumidores.

5.3 Considerações Finais

Para aplicações de *hard real time* é necessário que se forneçam garantias em tempo de projeto para suas restrições temporais. Entretanto, no que diz respeito a aplicações de *soft real time*, oferecer essas garantias pode ter um custo extremamente alto. Assim, em muitos casos opta-se por não oferecer garantias em tempo de projeto, podendo ocorrer situações de sobrecarga no sistema.

Em sistemas de tempo real não críticos as situações de sobrecarga, apesar de não serem desejáveis, são toleráveis. Porém, não ter nenhum controle sobre as condições do sistema nesses momentos pode ocasionar perdas ou degradações maiores que as necessárias.

Por este motivo, muitos mecanismos de adaptação foram e estão sendo apresentados na literatura para lidar melhor com as situações de sobrecarga, procurando ter um controle previsível da degradação do sistema, como mostrado no capítulo 2. Seguindo este caminho, foi apresentado neste capítulo um novo mecanismo de adaptação baseado em algumas importantes idéias encontradas na literatura.

Baseando-se no conceito de realimentação e sendo voltado principalmente para o controle de atividades periódicas, o mecanismo de adaptação proposto monitora o tempo de resposta das tarefas cadastradas. Ele atua alterando o período das mesmas, procurando manter o desempenho do sistema próximo ao patamar especificado (SDY).

Além do próprio mecanismo, foi apresentada uma implementação do mesmo utilizando as ferramentas do RT-CORBA e do ORB TAO. As tarefas periódicas foram implementadas como consumidores de eventos do tipo *push* e são ativadas em um POA configurado com um *ThreadPool*, tendo o número estático e dinâmico de *threads* definidos pelo programador. O mecanismo de adaptação foi implementado como um serviço, *AdaptiveService* – serviço de adaptação, o qual também é uma atividade periódica. O serviço de adaptação recebe as informações de tempo de resposta de cada tarefa periódica cadastrada no mesmo e atua nos momentos de sobrecarga de forma a manter o sistema próximo dos valores de controle desejáveis.

Com a proposta deste mecanismo deseja-se aumentar as possibilidades de situações em que os sistemas de tempo real continuarão executando corretamente. Apresentada esta proposta, segue no capítulo 6 a validação da mesma através da implementação de um sistema de tempo real de teste, e da realização de um conjunto de experiências, as quais procuram mostrar que o mecanismo proposto é capaz de manter um controle mais previsível no caso de sobrecarga.

6 Experiências

No capítulo 5 foi apresentado o mecanismo de adaptação proposto neste trabalho. Além dos seus princípios básicos foi apresentada uma implementação possível utilizando o ORB TAO.

O mecanismo proposto visa controlar a degradação do sistema na presença de sobrecarga. Procurando validar o mecanismo proposto, este capítulo tem por objetivo apresentar os resultados obtidos com as várias experiências realizadas sobre uma aplicação experimental que utiliza um protótipo do mecanismo de adaptação.

6.1 Estrutura da aplicação experimental

Pode-se encontrar inúmeras classes de aplicações de tempo real em ambiente distribuído que poderiam ser utilizadas como objeto de teste do mecanismo de adaptação proposto neste trabalho.

Um tipo de aplicação de tempo real que possui muitas possibilidades de estudo é a dos sistemas de automação industrial. Neste tipo de sistema podem ser encontradas tarefas periódicas, aperiódicas, esporádicas, tarefas ativas e reativas e tarefas com diferentes prioridades dentro do sistema.

Apesar dos sistemas de automação industrial normalmente serem considerados como aplicações de *hard real time* pode-se entender que o sistema aqui apresentado possui duas camadas de *software*, que podem ser vistas na FIGURA 6.1. A primeira, que pode estar localizada em controladores programáveis ou outros equipamentos possíveis de programação é responsável pelas atividades de mais alto risco, sendo esta camada de *software* considerada de *hard real time*. A segunda camada, de *soft real time*, é responsável por atividades de monitoração para efeitos de histórico ou de fornecimento de informações gerenciais e também atividades de controle que não necessitam da precisão temporal para estar na primeira camada. Esta camada é composta de tarefas que se comunicariam entre si e com a primeira camada de *software*.

Seguindo este caminho procurou-se neste trabalho identificar e representar alguns dos tipos de tarefas que podem ser encontrados neste tipo de sistemas, que são:

- **Sensores** e **Atuadores** foram os elementos da classe de aplicações de sistemas de automação escolhidos para representar as tarefas passivas. São derivados da classe *Task* e possuem métodos definidos em IDL, que podem ser invocados por elementos ativos do sistema. Seu comportamento interno simula o tempo de processamento que é cadastrado junto ao serviço de escalonamento na fase *offline* do escalonamento.
- Nos sistemas de automação foram identificadas três possíveis classes de tarefas periódicas, que são definidas como classes derivadas de *PeriodicTask*: (i) **Historico** representa as tarefas que são responsáveis pela coleta de dados de *log*, de modo a armazenar a situação do sistema ao longo do tempo, considerada uma tarefa de pouca prioridade e com um período razoavelmente alto (como um segundo); (ii) **Operador** representa aqueles processos de monitoramento (que realizam coleta e apresentação das informações) que têm interação com o operador do sistema; (iii) a classe **Alarme** representa

processos de monitoramento de informações que, em determinados casos poderiam identificar problemas no sistema e agir sobre o mesmo invocando atuadores caso necessário. Supõe-se que o período de tarefas da classe alarme seja menor que das outras tarefas a fim de estar sempre atualizado com os últimos dados coletados no sistema. Essas três tarefas também simulam um tempo de processamento que é informado na *RT_Info*.

O comportamento de cada classe definida acima, além de simular o tempo de execução determinado na fase *offline* da aplicação também reflete as dependências[7] definidas, seja *oneway* ou *twoway*. Os detalhes sobre o cadastro destas tarefas, suas restrições temporais e dependências pode ser visto na seção 6.2 deste trabalho.

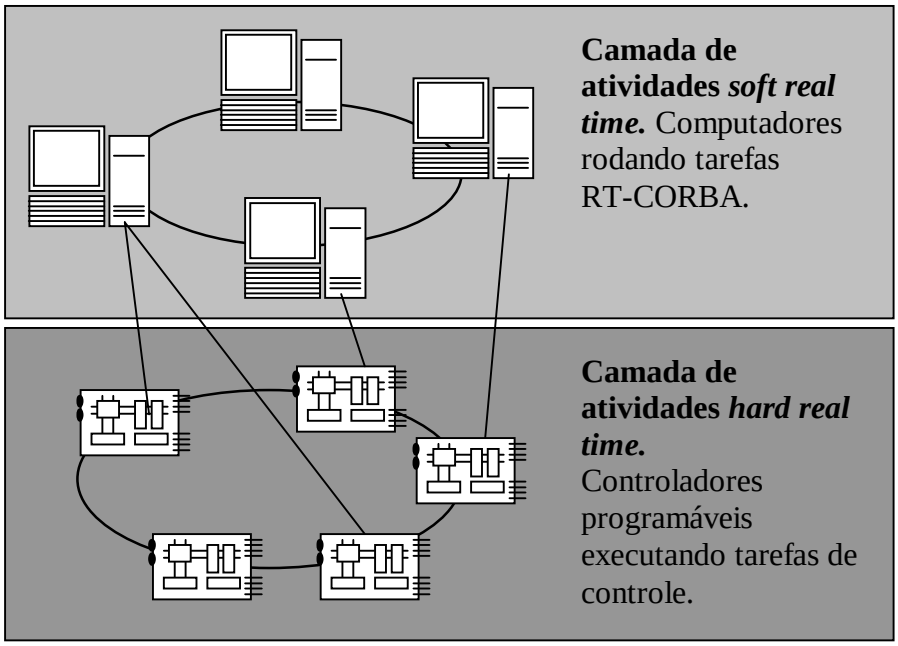


FIGURA 6.1 - Estrutura possível de um sistema de automação industrial.

6.2 Cenário das experiências

De modo a validar o mecanismo de adaptação proposto, foram realizados conjuntos de experimentos sobre um mesmo cenário de teste. Considerando a classe de aplicações escolhida como base de teste para o trabalho (automação industrial), onde em muitos casos a configuração inicial do sistema nunca é alterada, mantendo-se o mesmo número de tarefas e o mesmo número de equipamentos de *hardware* (sejam computadores, sensores, atuadores, etc.) ao longo do tempo, procurou-se da mesma forma, manter o mesmo cenário de teste e realizando alterações apenas no mecanismo de adaptação proposto.

[7] O tipo da dependência é referente à forma de invocação de método de um objeto. Dependência do tipo *oneway* se refere a invocações de métodos definidos com essa palavra chave em IDL, significando que o método não bloqueia o objeto chamador. No caso de dependência *twoway*, o método bloqueia o objeto chamador até o recebimento de uma resposta.

Assim, para todos os experimentos mostrados nesta seção, foi utilizada uma rede formada por duas máquinas, com configuração de *hardware* e *software* apresentada na TABELA 6.1.

TABELA 6.1 - Configuração de *hardware* e *software* do cenário das experiências.

Máquina	***Hardware***	***Software***
Nodo 1	- Processador *Pentium* III 650MHz - 128MB de memória RAM	- Sistema Operacional *Windows* 2000 *Advanced Server* - Compilador *Visual* C++ 6.0 - ACE versão 5.2 - TAO versão 1.2
Nodo 2	- Processador AMD K7 *Duron* 750MHz - 128MB de memória RAM	- Sistema Operacional *Windows* 2000 *Advanced Server* - Compilador *Visual* C++ 6.0 - ACE versão 5.2 - TAO versão 1.2

As tarefas foram distribuídas entre as máquinas de forma arbitrária e também estática. São inúmeras as possibilidades de atribuição das tarefas às máquinas, assim como são inúmeros os argumentos para a escolha de uma ou outra forma. Poderia-se utilizar uma atribuição de tarefas de forma que a carga se mantivesse homogeneamente distribuída entre as máquinas. Ainda, poder-se-ia utilizar critérios de afinidade entre as tarefas, como tarefas com uma grande comunicação entre si permanecerem juntas em uma mesma máquina. Entretanto, para uma aplicação de teste, todo o estudo necessário para a correta distribuição das tarefas pode ser deixado de lado em um primeiro momento, o que certamente não pode ser feito no caso de uma aplicação real.

Assim, a TABELA 6.2 mostra a alocação de tarefas utilizada nas experiências, bem como seus tempos de execução, período, dependências cadastradas no serviço de escalonamento e a prioridade calculada por este. Além disso, como o serviço de escalonamento do TAO não considera a presença de mais de um processador e foi utilizado apenas um serviço de escalonamento global, o resultado do escalonamento é indicado como utilização do processador excedida. Fato que não é necessariamente verdade pela presença de mais de um processador no sistema.

TABELA 6.2 - Alocação de tarefas entre os nodos da rede.

Nodos	**Tarefas**						
	Nome	**Pior Tempo de Execução**	**Período**	**Dependências**		**Prioridade**	
				oneway	***twoway***	**CORBA**[8]	**Win 2000**
1	sensorA	10 ms	-	-	-	2	1
	atuadorA	20 ms	-	-	-	0	15
	atuadorB	20 ms	-	-	-	0	15
	alarme1	20 ms	100 ms	atuadorA atuadorB	sensorA sensorB	0	15
	operador1	300 ms	500 ms	-	sensorA sensorB	1	2

[8] O valor zero representa maior prioridade e quanto maior o valor, menor é a prioridade da tarefa.

2	sensorB	10 ms	-	-	-	2	1
	alarme2	20 ms	100 ms	atuadorA atuadorB	sensorA sensorB	0	15
	operador2	300 ms	500 ms	-	sensorA sensorB	1	2
	historico1	500 ms	1000 ms	-	sensorA sensorB	2	1
	historico2	500 ms	1000 ms	-	sensorA sensorB	2	1
	Adaptive Service	40 ms	1000 ms	-	-	2	1
	Naming Service	-	-	-	-	-	-
	Event Service	-	-	-	-	-	-
	Schedule Service	-	-	-	-	-	-

6.3 Resultados obtidos

Cada execução da aplicação experimental tem uma duração de aproximadamente 2 minutos. As tarefas de menor período, alarme, recebem em torno de 1500 eventos periódicos, enquanto que as tarefas operador e histórico recebem aproximadamente 300 e 150 eventos respectivamente.

Inicialmente foram feitas execuções do sistema sem que o serviço de adaptação (*AdaptiveService*) atuasse de qualquer forma no sistema, mas apenas calculasse o DY(t) a cada período de sua execução. Assim, as TABELA 6.3 e TABELA 6.4 mostram as médias dos valores encontrados de período e atraso das tarefas periódicas para cada uma das 12 rodadas de execução da aplicação feitas. No final é apresentada a média desses valores. Já a FIGURA 6.2 mostra a evolução do DY(t) ao longo do tempo para a experiência 7 que está sinalizada nas tabelas. Todos os valores de tempo das tabelas têm como unidade milissegundos (ms) e, os valores negativos nas colunas de média dos atrasos indica sobra de tempo.

TABELA 6.3 - Média dos valores de período e atraso do sistema sem adaptação (i).

Rodadas	**alarme1 [ms]**		**operador1 [ms]**		**historico1 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	106,81	663,54	500,02	-273,94	1000,3	-701,95
2	105,91	1256,21	499,98	-208	1000,35	-790,74
3	104,59	379,96	500,08	-241,39	1000,24	-777,59
4	104,74	3081,39	500	-252,8	1000,33	-798,53
5	104,99	1790,77	499,98	-258,34	1000,18	-746,13
6	106,66	341,84	499,98	-252,87	1000,28	-797,58
7	103,91	1001,78	500,11	-229,53	1000,15	-781
8	103,82	1378,18	500,41	-201,07	1000,25	-797,77
9	104,74	750,06	500,05	-191,88	1000,43	-795,47
10	102,08	235,18	500,22	-241,25	1000,17	-788,78

11	103,25	815,89	500,05	-218,22	1000,67	-799,87
12	103,49	3285,3	500,05	-199,81	1000,25	-791,74
Média	**104,58**	**1248,34**	**500,07**	**-230,78**	**1000,3**	**-780,59**
Desvio	**1,339**	**968,711**	**0,119**	**25,555**	**0,135**	**27,649**

TABELA 6.4 - Média dos valores de período e atraso do sistema sem adaptação (ii).

Rodadas	**alarme2 [ms]**		**operador2 [ms]**		**historico2 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	106,09	700,42	500,07	-326,87	1000,12	-628,5
2	101,84	192,61	500,08	-353,15	1000,82	-740,75
3	106,42	123,96	500,07	-353,59	1001,29	-740,03
4	102,34	114,15	500,1	-354,03	1000,15	-750,1
5	102,92	317,66	500,07	-348,55	1000,67	-684,04
6	101,59	167,3	500,08	-356,21	1001,27	-763,7
7	103,67	253,16	500,13	-340,62	1001,1	-726,65
8	102,34	196,27	500,09	-354,35	1000,72	-749,7
9	101,84	254,96	500,07	-352,91	1000,19	-760,73
10	103,34	109,74	500,07	-322,93	1000,68	-758,5
11	102,67	149,1	500,07	-357,39	1000,47	-751,97
12	103,67	139,47	500,07	-356,86	1001,14	-733,47
Média	**103,14**	**226,56**	**500,08**	**-348,12**	**1000,71**	**-732,34**
Desvio	**1,348**	**155,545**	**0,017**	**11,262**	**0,407**	**37,366**

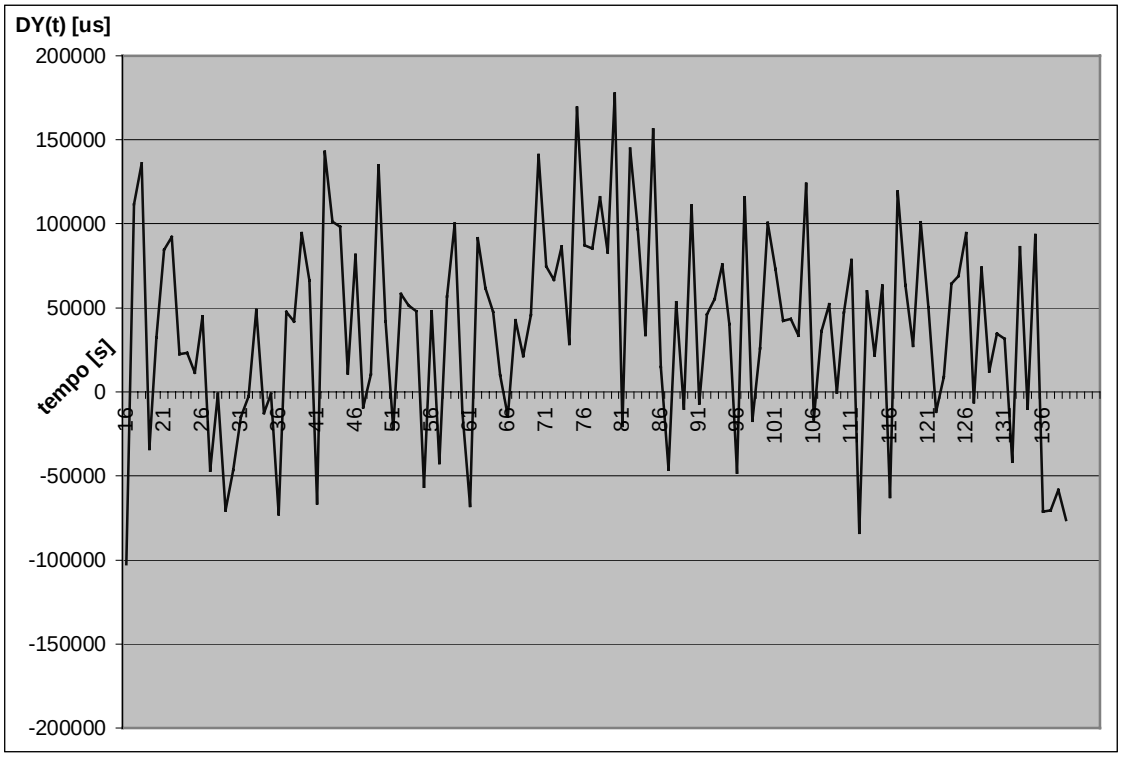


FIGURA 6.2 - Evolução do DY(t) ao longo do tempo para o sistema sem adaptação.

Pela análise das TABELA 6.3, TABELA 6.4 e FIGURA 6.2 se observa que ocorrem atrasos significativos afetando com mais rigor as duas tarefas de maior prioridade (alame1 e alarme2). Apesar de possuir a maior prioridade, esses processos são também os mais exigentes em termos de ocupação de processador, acabam sofrendo atrasos por este motivo, pela própria característica do *Windows* de não ser um sistema

operacional preemptivo por prioridade puro e também pela característica do protocolo TCP/IP de apresentar filas de envio baseado na política de FIFO.

Em um primeiro momento foi implementada uma versão ainda grosseira de um algoritmo de adaptação. Neste algoritmo, em cada ativação do serviço de adaptação (foi utilizado o período de um segundo) é calculado o valor atual de DY(t) e então comparado com o valor de SDY (valor que se assumiu como sendo zero). Caso DY(t) fosse maior que SDY então o serviço de adaptação atua no sistema incrementando em um o fator de multiplicação dos períodos das tarefas e então atualizando no serviço de eventos o período de cada tarefa cadastrada no serviço de adaptação. Caso DY(t) seja menor que SDY por dois ou três ciclos consecutivos (valor configurado pelo programador), o fator de multiplicação dos períodos das tarefas é decrementado de um (contanto que seja sempre maior ou igual a um) e então os períodos são atualizados no serviço de eventos. Os resultados encontrados utilizando-se esta primeira versão de adaptação podem ser vistos em TABELA 6.5, TABELA 6.6 e FIGURA 6.3.

TABELA 6.5 - Média dos valores de período e atraso do sistema com a primeira forma de adaptação (i).

Rodadas	**alarme1 [ms]**		**operador1 [ms]**		**historico1 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	161,84	48,69	879,78	-195,86	1868,54	-750,68
2	154,47	62,83	830,56	-187,08	1743,69	-757,4
3	142,99	44,83	784,97	-346,69	1527,22	-653,74
4	147,09	43,28	807,52	-349,28	1568,73	-664,88
5	149,01	45,45	820,15	-339,37	1602,01	-654,46
6	149,61	38,37	819,28	-340,93	1643,69	-656,14
7	139,57	50,04	760,24	-345,86	1518,73	-723,92
8	151,4	42,45	839,64	-347,28	1668,7	-658,78
9	148,14	42,47	801,17	-346,51	1560,38	-649,1
10	151,42	39,59	823,83	-339,97	1652,03	-649,75
11	147,54	39,77	811,36	-349,55	1610,34	-628,1
12	145,45	45,89	787,57	-341,73	1527,07	-614,73
Média	**149,04**	**43,3**	**813,83**	**-319,17**	**1624,26**	**-671,73**
Desvio	**5,407**	**6,280**	**28,962**	**57,237**	**97,857**	**44,275**

TABELA 6.6 - Média dos valores de período e atraso do sistema com a primeira forma de adaptação (ii).

Rodadas	**alarme2 [ms]**		**operador2 [ms]**		**historico2 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	160,16	103,71	903,76	-350,58	1867,87	-635,73
2	154,49	49,87	870,32	-347,84	1742,98	-638,77
3	142,34	50,12	764,15	-292,91	1518,15	-557,61
4	145,5	56,47	799,59	-284,48	1568,14	-547,32
5	147,43	52,11	801,68	-265,48	1601,36	-550,29
6	147,84	44,95	814,11	-286,88	1609,71	-544,89
7	139,03	36,72	753,69	-354,64	1493,6	-592,45
8	149,11	43,33	826,59	-277,57	1668	-528,02
9	146,08	51,69	789,16	-276,74	1551,39	-533,19
10	150,25	54,27	820,35	-279,02	1626,27	-532,9
11	146,09	52,81	787,03	-287,74	1609,63	-551,58
12	144,04	54,54	774,59	-276,87	1526,41	-587,75
Média	**147,69**	**54,21**	**808,75**	**-298,39**	**1615,29**	**-566,7**
Desvio	**5,301**	**15,847**	**41,308**	**31,126**	**100,709**	**36,808**

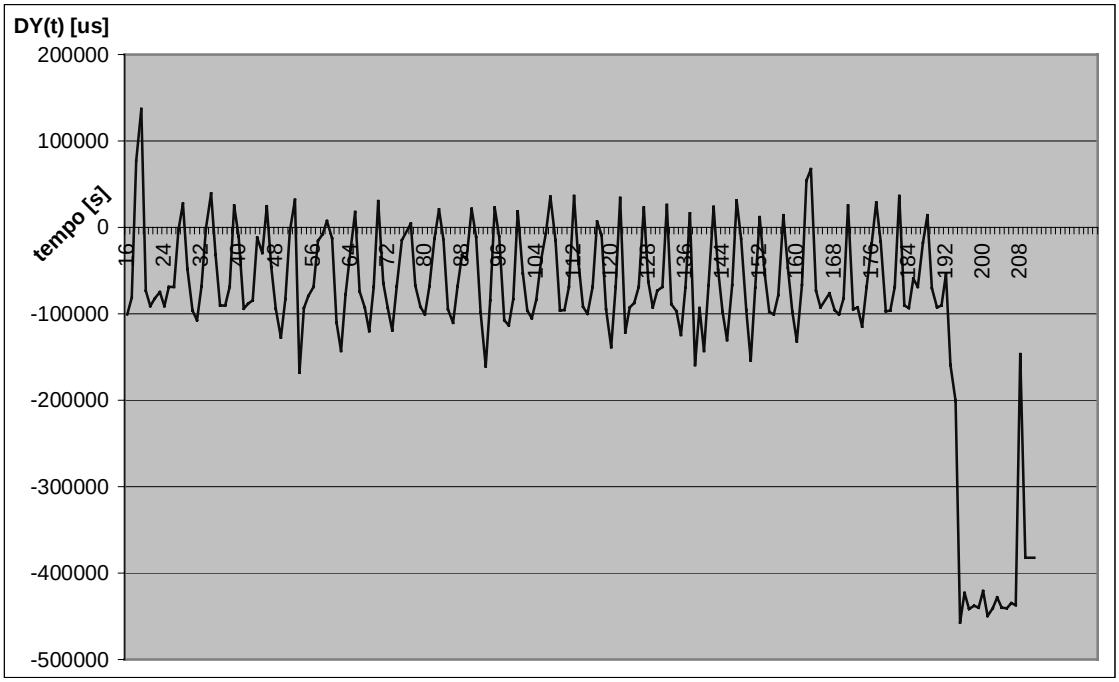


FIGURA 6.3 - Evolução do DY(t) ao longo do tempo com a primeira forma de adaptação (i).

A partir do instante em que algumas tarefas param de executar, o sistema passa a apresentar tempos de atraso menores, momento no qual o serviço de adaptação passa a não atuar, ou atuar menos no sistema. Isto ocorre, na FIGURA 6.3, por volta do instante 196. Portanto, este momento de término das tarefas não será mais mostrado nos demais gráficos e, para melhor comparação com o gráfico da experiência sem adaptação, a evolução de DY(t) na primeira forma de adaptação aparece novamente na FIGURA 6.4 na mesma escala que naquele caso.

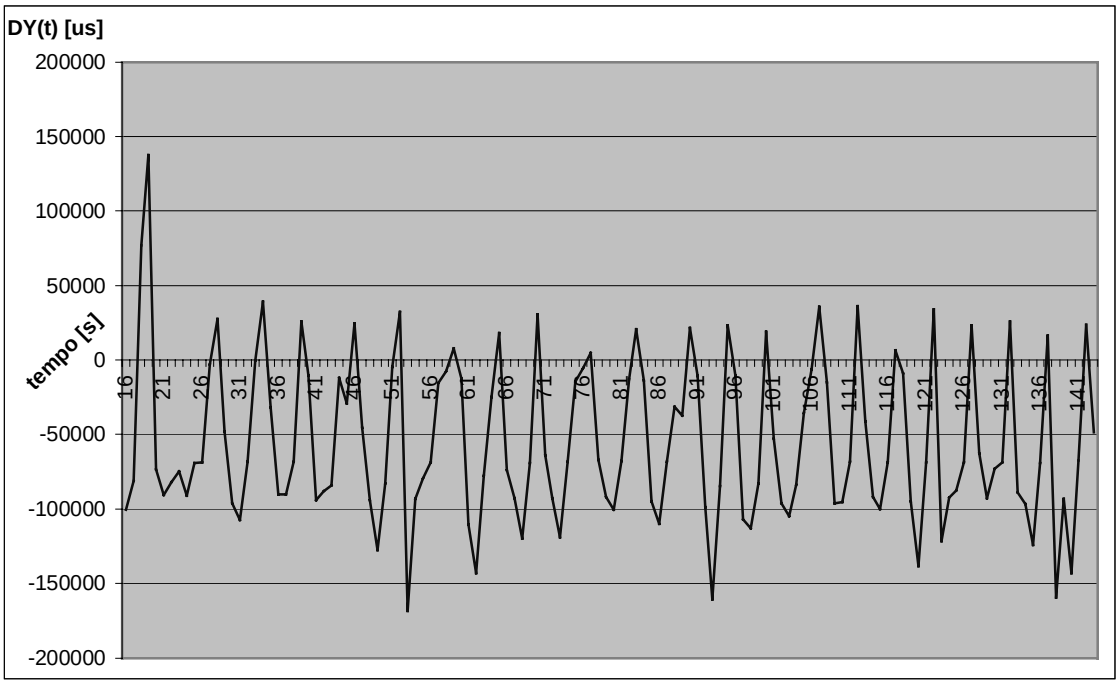


FIGURA 6.4 - Evolução do DY(t) ao longo do tempo com a primeira forma de adaptação (ii).

Com esta primeira versão do mecanismo de adaptação, o sistema apresentou uma melhora significativa nos tempos de média de atraso. Entretanto, pela visualização do gráfico de evolução de DY(t) pode-se notar que a adaptação ainda insere instabilidade no sistema. Isto se deve a sua forma de atuação ser ainda excessivamente forte mesmo com pequenos índices de atraso. Além disso, os valores médios de atraso ainda mostram perdas nas tarefas de menor período, mesmo com a atuação. Assim, buscando-se um ajuste mais fino da adaptação, foi utilizado o mecanismo de adaptação descrito na seção 5.1 no restante das experiências.

6.3.1 Experiências preliminares com o mecanismo de adaptação proposto

Um dos primeiros testes realizados foi utilizando-se K_P=1/250000 e SDY=-50000 us. Os resultados podem ser vistos na FIGURA 6.5 e TABELA 6.7. Observa-se que os tempos de atraso são um pouco menores (mas ainda existentes) que aqueles encontrados com a primeira versão da adaptação, mas que a adaptação continua um pouco instável, com oscilações em torno do valor determinado para SDY. Além disso, nota-se que os atrasos diminuíram mesmo com as tarefas (como alarme1 e alarme2) executando com períodos médios menores (de aproximadamente 149ms, na primeira versão, para aproximadamente 115ms nas tarefas alarme1 e alarme2).

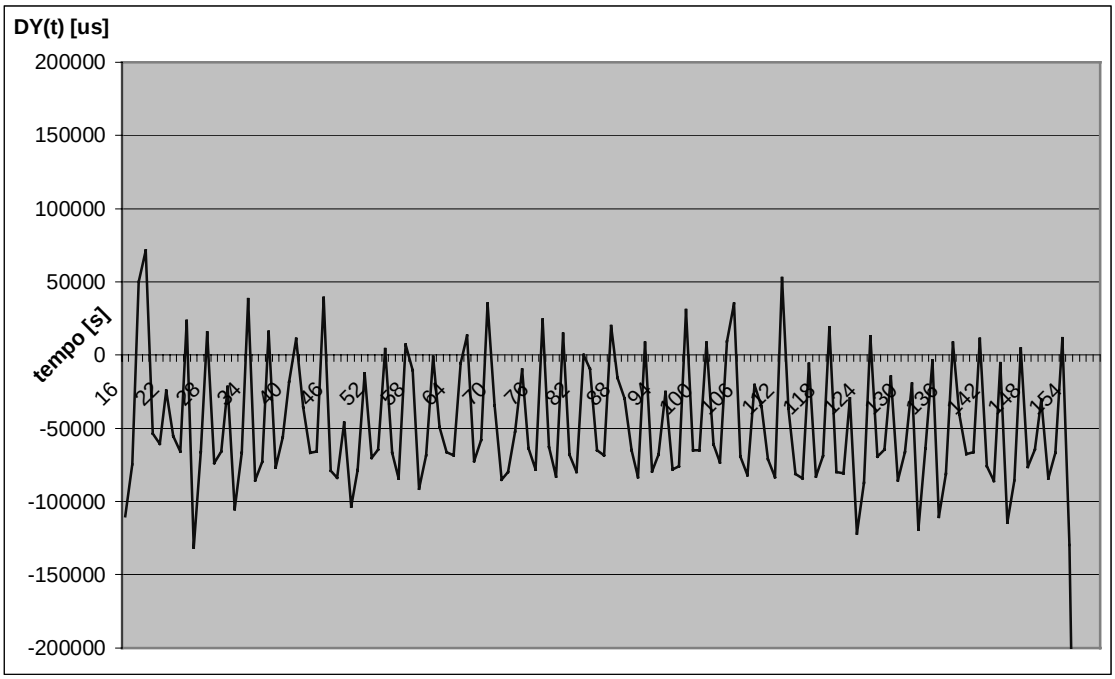


FIGURA 6.5 - Evolução DY(t) para adaptação com SDY=-50000us e K_P=1/250000.

Após este primeiro teste foram realizadas outras experiências com o objetivo de analisar a atuação do mecanismo em relação aos tempos de atraso das tarefas e da sua estabilidade. Em alguns dos testes realizados notou-se um comportamento não esperado do sistema, muito provavelmente devido ao serviço de eventos de tempo real do TAO não estar totalmente preparado para o tipo de uso que foi feito no protótipo apresentado.

A aplicação experimental que tem sua duração aproximada de 2 minutos foi dividida em duas fases. As tarefas começam a receber eventos assim que se conectam

com o serviço de eventos, e a primeira fase inicia no que se denominou de *"prepare time"* no qual as tarefas contatam o serviço de nomes e obtêm referências para os objetos que necessitam acessar (sensores e atuadores). Após esta fase de preparação, no instante chamado de *"start time"*, as tarefas passam a efetivamente executar e enviar os dados para o serviço de adaptação.

TABELA 6.7 - Médias de período e atraso para adaptação com SDY=-50000us e K_P=1/250000.

Tarefa	Média do período (P)	Média do atraso (R-D)
alarme1	115,82 ms	34,50 ms
alarme2	115,47 ms	24,78 ms
operador1	687,87 ms	-348,11 ms
operador2	687,16 ms	-321,7 ms
historico1	1668,80 ms	-713,25 ms
historico2	1659,74 ms	-640,37 ms

Em alguns dos casos de teste observou-se um comportamento inesperado logo após o tempo de preparação. No primeiro evento em que o serviço de adaptação recebeu informações das tarefas, ele calculou os novos períodos efetivos e atuou no sistema aparentemente de forma correta (os valores de períodos analisados na depuração do sistema se mostraram corretos), mas a resposta para a alteração dos períodos para duas das tarefas do sistema (historico1 e historico2) não foi correta. Enquanto as outras tarefas tiveram os seus períodos alterados conforme o solicitado pelo serviço de adaptação, essas duas tarefas tiveram seu recebimento de eventos como que cancelado por um período em torno de 130s ou menos em determinados casos. Após esse tempo sem receber eventos essas tarefas passaram a receber seus eventos no período normal para o qual elas haviam sido cadastradas.

A TABELA 6.8 apresenta uma relação de algumas das experiências realizadas e informações qualitativas sobre seu comportamento.

TABELA 6.8 - Relação de algumas das experiências realizadas.

SDY	K_P	Comportamento
-50000us	1/250000	Sistema executou corretamente apresentando resultados um pouco melhores que a primeira forma de adaptação.
-50000us	1/300000	Executou corretamente, resultados similares ao anterior.
-50000us	1/400000	Executou corretamente, sem diferenças significativas em relação ao caso anterior.
-60000us	1/300000	Executou corretamente, sem diferenças significativas em relação ao caso anterior.
-80000us	1/480000	Apresentou o comportamento indesejado descrito.
-100000us	1/250000	Apresentou o comportamento indesejado descrito.
-100000us	1/300000	Apresentou o comportamento indesejado descrito.

A partir destas experiências observou-se que o comportamento inesperado descrito anteriormente tornou-se constante quando se assumia o valor de SDY menor ou igual a –80000us. Devido a restrições temporais não foi possível encontrar a origem do problema de forma segura. Entretanto, apesar de restringir as possibilidades de configurações do mecanismo de adaptação o problema encontrado não chega a invalidar o protótipo apresentado.

6.3.2 Resultados com o mecanismo de adaptação proposto

A estabilidade do controle feito pelo mecanismo de adaptação depende dos valores associados à K_P e SDY. O ajuste destas constantes poderia ser feito através de cálculos matemáticos da área de engenharia de controle, relativos à política de controle proporcional. Porém, não era objetivo deste trabalho aprofundar o estudo nesta área. Desta forma, procurou-se, através de experimentação (como visto na seção 6.3.1), alterar os valores das constantes K_P e SDY de modo que a atuação no sistema se tornasse mais suave e ainda que, no caso da existência de pequenas oscilações, elas se mantivessem em valores negativos de DY(t).

Após estes testes chegou-se aos valores de K_P=1/420000 e SDY=-60000us, cujos resultados podem ser vistos nas TABELA 6.9,TABELA 6.10 e FIGURA 6.6.

TABELA 6.9 - Médias dos tempos na primeira rodada com a segunda versão de adaptação (i).

Rodadas	**alarme1 [ms]**		**operador1 [ms]**		**historico1 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	115,83	-8,86	736,58	-357,31	1718,71	-707,22
2	115,73	47,27	687,07	-255,23	1760,29	-758,73
3	115,1	37,01	679,07	-253,65	1685,58	-759,7
4	114,26	-9,24	731,45	-359,57	1760,53	-697,25
5	115,25	-5,06	749,18	-361,95	1818,71	-712,15
6	113,52	-17,22	762,7	-363,43	1918,61	-750,64
7	114,18	-12,58	747,17	-365,11	1818,7	-715,1
8	112,28	27,9	664,36	-349,56	1593,77	-738,42
9	112,55	28,74	664,31	-345,76	1627,07	-741,55
10	112,29	25,44	666,44	-348,92	1577,15	-758,48
11	112,32	20,72	676,83	-356,23	1635,25	-739,35
12	112,83	-16,32	750,85	-361,49	1885,31	-729,9
Média	**113,84**	**9,81**	**709,66**	**-339,85**	**1733,3**	**-734,04**
Desvio	**1,333**	**22,433**	**37,835**	**38,638**	**108,29**	**20,761**

TABELA 6.10 - Médias dos tempos na primeira rodada com a segunda versão de adaptação (ii).

Rodadas	**alarme2 [ms]**		**operador2 [ms]**		**historico2 [ms]**	
	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)	Média dos Períodos (P)	Média dos Atrasos (R-D)
1	112,83	-0,94	726,68	-307,67	1659,7	-623,77
2	114,32	37,49	707,9	-363,2	1752,4	-673,78
3	113,84	46,14	689,12	-362,17	1701,6	-688,84
4	112,92	-2,82	722,47	-313,38	1752,81	-626,55
5	113,25	-2,67	734,99	-304,42	1793,75	-653,25
6	112,19	-14,28	757,84	-329,55	1901,27	-653,62
7	112	-3,67	737,07	-315,83	1809,65	-625,84
8	112,21	8,94	659,95	-367,51	1584,65	-665,64
9	112,31	16,2	666,25	-356,36	1609,7	-644,61
10	111,25	15,24	655,89	-358,74	1576,35	-650,28
11	110,98	5,91	672,47	-371,16	1626,24	-639,55
12	111,27	-9,71	739,12	-329,9	1876,23	-642,69
Média	**112,44**	**7,98**	**705,81**	**-339,99**	**1720,36**	**-649,03**
Desvio	**0,990**	**17,574**	**34,050**	**24,480**	**106,752**	**19,027**

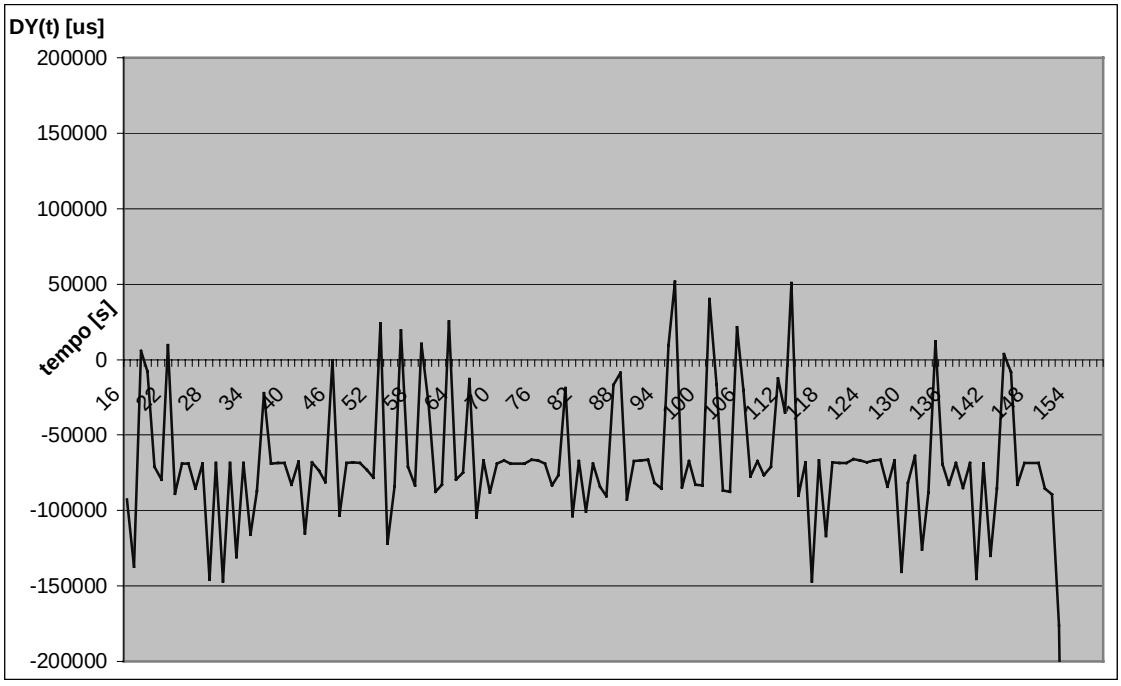


FIGURA 6.6 - Evoluação de DY(t) com SDY=-60000us e K_P=1/420000.

Observa-se na FIGURA 6.6 que há uma estabilidade um pouco maior da atuação sobre o sistema. Após o pico inicial de carga o sistema atua de forma a melhorar o DY(t) do sistema. O deslocamento de SDY para um valor mais negativo (no caso, -70000us) teve, como conseqüência, a manutenção do valor de DY(t) próximo deste valor devido à atuação do serviço de adaptação. Então, as médias dos atrasos (observados nas TABELA 6.9 e TABELA 6.10) se mantiveram negativas ou com um valor baixo para todas as tarefas, significando que, com pequenas alterações nos períodos (os quais ficaram, em sua maioria, até abaixo dos períodos médios resultantes mostrados na TABELA 6.7) das tarefas pelo serviço de adaptação, todas conseguiram manter-se próximas dos valores seus *deadlines*.

TABELA 6.11 - Tabela comparativa entre o sistema sem adaptação e com adaptação.

Tarefas	**Sistema sem adaptação [ms]**		**Sistema com adaptação (SDY=-60000us e K_P=1/420000) [ms]**	
	Média dos Períodos (P)	**Média dos atrasos (R-D)**	**Média dos Períodos (P)**	**Média dos atrasos (R-D)**
alarme1	104,58	1248,34	113,84	9,81
alarme2	103,14	226,56	112,44	7,98
operador1	500,07	-230,78	709,66	-339,85
operador2	500,08	-348,12	705,81	-339,99
historico1	1000,3	-780,59	1733,3	-734,04
historico2	1000,71	-732,34	1720,36	-649,03

Finalmente, a TABELA 6.11 mostra os resultados obtidos com o sistema sem adaptação e com a melhor configuração de SDY e K_P encontradas. O objetivo desta tabela é dispor os dados lado a lado para facilitar a comparação entre as duas situações.

6.4 Considerações Finais

Este capítulo teve como objetivo apresentar um protótipo do mecanismo de adaptação proposto em funcionamento. Assumiu-se como pano de fundo um sistema de automação industrial hipotético, dividido em duas camadas, uma de *hard real time* (onde as garantias seriam ainda em tempo de projeto e não haveria atuação do mecanismo) e outra *soft real time* cujo controle em caso de sobrecarga estaria a cargo do mecanismo de adaptação.

A plataforma alvo foi um sistema operacional de propósito geral e a rede *Ethernet*, que não ofereciam certas garantias necessárias para sistemas de tempo real. Desta forma, a sobrecarga à qual o sistema sofre (e necessária para o teste do mecanismo) não precisaria vir de cargas artificialmente induzidas no sistema, mas sim das próprias variações do sistema operacional e da rede.

Os resultados tabelados para o sistema executando sem o mecanismo de adaptação mostraram a sua imprevisibilidade quando expostos às flutuações citadas. A utilização de um mecanismo de adaptação mesmo que grosseiro (como mostrados nas TABELA 6.5 e TABELA 6.6) se mostrou benéfica no controle do sistema em caso de sobrecarga. Os resultados de períodos e atrasos se mantiveram mais próximos do desejado.

Várias experiências preliminares foram feitas com o mecanismo de adaptação para encontrar valores de SDY e K_P que apresentassem um resultado melhor no sistema. Esses valores vão depender muito das condições de cada aplicação. Como afirmado anteriormente, é necessário que o engenheiro de controle ou o projetista do sistema determine os valores mais adequados para esses parâmetros conforme cálculos matemáticos ou medições experimentais feitos sobre a sua aplicação específica.

Mesmo com problemas no protótipo (muito possivelmente causados pela utilização do serviço de eventos em condições não previstas por este), os resultados apresentados nesta seção mostram que, mesmo lidando com um conjunto de tarefas apenas representativas, o mecanismo de adaptação proposto pode ser utilizado em casos reais com as referidas configurações necessárias para cada aplicação.

7 Conclusões e Trabalhos Futuros

O desenvolvimento da computação distribuída teve um grande crescimento não só ligado ao aumento do número de aplicações nessa área, mas também ao aumento da oferta de ferramentas e propostas para melhorar e facilitar o desenvolvimento de aplicações distribuídas. Seguindo essa tendência, a OMG buscou a unificação das diversas tecnologias surgidas em um único padrão, o CORBA.

O CORBA surgiu, portanto, como a solução única para os problemas da computação distribuída. Na verdade, tornou-se mais um dos padrões existentes no mercado, ao lado de exemplos como DCOM e .NET (iniciativas de objetos distribuídos da *Microsoft*) e *Jini* (rede de Serviços da *Sun*), para facilitar, integrar e reutilizar objetos em sistemas abertos em ambiente distribuído.

A escolha da OMG de não criar implementações, mas apenas definições de mecanismos para o CORBA e seus serviços trazem, por vezes, problemas, principalmente no que se refere à compatibilidade entre as diversas implementações de CORBA do mercado. Da mesma forma, a variedade de implementações do CORBA tanto provindas do mercado corporativo quanto do meio acadêmico têm influído na maior aceitação do CORBA como o *middleware* para ambientes heterogêneos distribuídos. Aliado a sua independência de plataforma e principalmente linguagem, o que permite a integração de sistemas legados a novos sistemas distribuídos, o CORBA tem sido utilizado em várias aplicações de mercado.

O CORBA trouxe vantagens para o desenvolvimento de aplicações distribuídas de modo que começou a ser utilizado também para aplicações de tempo real. Porém, apresenta deficiências para este tipo de aplicação que impulsionaram a definição de uma extensão específica para este fim, o RT-CORBA.

O RT-CORBA é ainda um padrão em desenvolvimento, foram criados mecanismos para deixar os ORBs mais determinísticos, para aumentar desempenho e para permitir a especificação e obtenção de garantias de tempo real. Nisto reside um dos problemas e também um dos benefícios da especificação. Suas definições são ao mesmo tempo flexíveis o suficiente para permitir a execução e definição de uma variedade de modelos de sistemas de tempo real, mas também não define políticas específicas, deixando a cargo das implementações de ORBs, ou até mesmo às aplicações a responsabilidade da definição de políticas específicas para cada caso.

As implementações existentes tentam seguir suas recomendações, mas em certos momentos acabam por criar soluções próprias para tornar seus ORBs mais determinísticos e com características a mais para a criação de aplicações de tempo real. Mesmo considerando implementações como o TAO, uma das mais completas de RT-CORBA encontradas e que ainda acrescenta melhorias, alguns pontos passam em branco.

O aspecto mais importante a ser levado em consideração, é que o CORBA e sua extensão RT-CORBA ainda são um *middleware*, e por este motivo dependem muito do sistema operacional e do *hardware* sobre o qual são colocados a executar. Por este motivo, por mais que o RT-CORBA (ou, mais especificamente, uma implementação como o TAO) se aproxime e solucione suas deficiências de definição e garantia de tempo e QoS, estruturas e forma de programação para tempo real, ele ainda é dependente e precisa que tanto o sistema operacional, *hardware*, quanto a estrutura de rede, suporte as exigências de tempo e qualidade de serviço que as aplicações de tempo real necessitam.

Conseqüentemente, em um ambiente que se pode não ter certeza sobre que tipo de suporte (seja *hardware* ou sistema operacional) que a aplicação terá que contar, abordagens utilizadas para sistemas de *hard real time* podem não ser as mais indicadas. Atividades neste tipo de ambiente não têm totais garantias de que terão todos os seus *deadlines* atendidos. Desta forma, é mais interessante considerar que esses sistemas abrigarão aplicações de *soft real time*, de modo que possam ser aplicadas abordagens de melhor esforço no intuito de ampliar a gama de situações em que a aplicação tenha suas restrições temporais atendidas.

Assim, pensando nesses aspectos que podem afetar a qualidade de uma aplicação de tempo real, e se preocupando especialmente com aplicações que suportassem uma abordagem de melhor esforço, foi apresentado neste trabalho um mecanismo de adaptação para o ambiente RT-CORBA.

O foco principal de preocupação foi com as tarefas periódicas que possuem restrições temporais. Embora a abordagem apresentada não seja possível em absolutamente todas as aplicações, ela pode ser usada sempre que existir alguma flexibilidade no período das tarefas. A atuação é baseada em um laço de realimentação que fornece as informações de tempo de resposta em uma ativação de cada tarefa para um serviço de adaptação. Escolheu-se como alvo da adaptação o período das tarefas. A variação do período é derivada do cálculo do que se chamou de DY(t), *delay profile*, o qual é, em última instância, a medida de quanto as tarefas perderam ou ganharam em relação ao seu *deadline* (período) em uma janela específica de medição (DW, *delay window*).

A idéia do mecanismo apresentado neste artigo é basicamente sacrificar o período das tarefas para manter sob controle os atrasos, ao mesmo tempo em que exerce controle sobre o comportamento do sistema em sobrecarga. Ao contrário da maioria dos sistemas, onde a sobrecarga gera uma degradação aleatória, a abordagem proposta permite uma degradação controlada nos momentos de sobrecarga. É também importante notar que a adaptação é dinâmica, capaz de responder às flutuações de carga que ocorram durante a execução do sistema. O mecanismo automaticamente obtém do sistema a maior qualidade possível, porém respeitando os *deadlines* das tarefas.

A validação do mecanismo proposto foi feita através da sua implementação utilizando os mecanismos do RT-CORBA e os serviços de escalonamento e eventos que aparecem, de forma diferenciada, no ORB TAO. Apesar da implementação do protótipo apresentar para certas situações um comportamento inesperado, os resultados mostram que para uma aplicação exemplo o mecanismo conseguiu melhorar a qualidade do sistema representada pela comparação entre o tempo de resposta e o *deadline* das tarefas. A presença da adaptação permitiu reduzir os atrasos das tarefas em função do não determinismo do sistema.

O mecanismo de adaptação utilizado atua de forma homogênea sobre as tarefas do sistema e é utilizado o mesmo *fator* de adaptação ao período das tarefas. Melhorias poderiam ser obtidas optando-se pelo cálculo de um fator de adaptação específico para cada tarefa, levando em consideração sua importância. Ainda, foi escolhido o mínimo múltiplo comum (MMC) do período das tarefas como o período do serviço de adaptação de forma *ad-hoc*, assim como acontece em outros trabalhos na literatura. É uma questão em aberto se esta escolha é a melhor para qualquer aplicação. Experiências para verificar esses aspectos e analisar a escalabilidade da solução, visto que os testes realizados para esse trabalho utilizaram apenas duas máquinas, estão entre alguns dos possíveis trabalhos futuros.

Além disso, como afirmado anteriormente, muito da previsibilidade que o RT-CORBA pode trazer para os sistemas de tempo real depende do suporte que lhe é

fornecido. As experiências foram realizadas sobre máquinas (*hardware*, rede) e sistema operacional de propósito geral. Melhores resultados poderiam ser obtidos tanto do ORB utilizado, o TAO, quanto do próprio mecanismo de adaptação apresentado neste trabalho, se pudessem ser realizadas experiências em um ambiente mais específico para aplicações de tempo real. Porém, o fato de se ter utilizado um suporte de propósito geral reforça a capacidade do mecanismo de adaptar a aplicação ao ambiente um tanto adverso no qual foi inserida. Mesmo sofrendo com as imprevisibilidades do suporte foram obtidos valores de atraso próximos do desejado.

Ainda, outras possibilidades de implementação do mecanismo proposto poderiam ser testadas. Também, uma substituição do serviço de eventos do TAO por outro que privilegiasse especificamente a geração de eventos periódicos e a reconexão dos consumidores poderia ser proposta e validada como uma alternativa para tornar mais rápida a resposta da adaptação.

Anexo 1 Instalação do ACE/TAO

Este anexo tem por objetivo traçar um roteiro de instalação da plataforma ACE/TAO com o intuito de que as experiências mostradas neste trabalho possam ser refeitas nas mesmas condições em que foram realizadas para este trabalho. Assim, cabe ressaltar que este breve manual de instalação não procura de forma alguma ser um "manual de instalação da plataforma ACE e do ORB TAO". Serão detalhados os passos de instalação e apenas os parâmetros de configuração utilizados para este trabalho, desconsiderando as várias possibilidades diferentes de configuração deste software.

Além dos mecanismos básicos do ORB, a implementação do mecanismo de adaptação apresentado neste trabalho utilizou os serviços de nomes, escalonamento e eventos do TAO. Desta forma, são mostrados os passos para instalação do ACE (necessário para a execução do TAO), do TAO propriamente dito e dos serviços citados anteriormente.

Os passos mostrados a seguir se referem, portanto, à instalação do ACE/TAO no ambiente *Windows* NT/2000 e considerando apenas aspectos relevantes às experiências realizadas neste trabalho:

1. Obtenção dos pacotes:

Este trabalho utilizou a versão 5.2 do ACE e 1.2 do TAO. Versões mais antigas que essas podem gerar erros não previstos na compilação ou execução da aplicação de teste criada. Mudanças em versões mais recentes podem gerar problemas de incompatibilidades, portanto, para a repetição das experiências realizadas sugere-se a utilização das mesmas versões. Novas versões destes pacotes podem ser encontradas em: http://deuce.doc.wustl.edu/Download.html. Deve ser copiado o arquivo denominado ACE+TAO.zip no caso da plataforma *Windows*.

2. Descompactação:

O arquivo ZIP copiado deve ser descompactado em um diretório local de forma que os nomes dos diretórios sejam preservados (esta opção do software de descompactação deve estar marcada). Supondo que o pacote tenha sido descompactado no *drive* C denomina-se:

ACE_ROOT – C:\ACE_wrappers

TAO_ROOT – C:\ACE_wrappers\TAO

Essas variáveis devem ser cadastradas como variáveis de ambiente do *Windows 2000* da seguinte forma:

- Clicar com o botão direito sobre o ícone *"My Computer"* e escolher a opção *"Properties"*;
- Na aba *"Advanced"* clicar no botão *"Enrivonment Variables..."*;
- No quadro *"User variables for..."* clicar no botão *"New"*;
- Na nova janela preencher o campo *"Variable Name"* com ACE_ROOT e o campo *"Variable Value"* com C:\ACE_wrapper;
- Repetir os dois últimos passos para a variável TAO_ROOT.

3. Configuração:

Antes de iniciar a compilação do pacote é necessário que sejam feitas configurações no ACE (para especificar que a plataforma é *Windows* e que será utilizada a política de escalonamento *Rate Monotonic*) e no serviço de eventos. Para

configurar o ACE deve-se criar um arquivo denominado "***config.h***" no diretório "**ACE_ROOT\ace"** com o seguinte conteúdo:

```
#include "ace/config-win32.h
#define TAO_USES_STRATEGY_SCHEDULER
#define TAO_USES_RMS_SCHEDULING
```

Para configurar o serviço de eventos para o correto funcionamento das experiências apresentadas neste trabalho deve-se alterar o arquivo ***"EC_defaults.h"*** do diretório **"TAO_ROOT\orbsvcs\orbsvcs\Event"** alterando o valor das constantes (*defines*) **TAO_EC_DEFAULT_CONSUMER_RECONNECT** e **TAO_EC_DEFAULT_DISCONNECT_CALLBACKS** para **1**.

4. Compilação:

Finalmente, após a configuração necessária pode-se iniciar a fase de compilação do pacote. O compilador utilizado para as experiências deste trabalho foi o *Microsoft Visual C++ 6.0*, portanto sugere-se a utilização deste mesmo compilador. Podem ser utilizadas outras versões ou mesmo versões do *Borland C++*, porém não há garantias sobre o correto funcionamento das experiências.

Para iniciar o processo de compilação deve-se carregar o arquivo de projeto ***"TAOACE.dsw"*** contido no diretório **TAO_ROOT**. A ordem de compilação dos projetos foi a seguinte:

- ACE DLL
- TAO DLL
- TAO IDL Compiler
- RTCORBA
- Naming_Service
- Scheduling_Service
- Event_Service

Anexo 2 Instalação da Aplicação Experimental

Ainda com o objetivo de facilitar a repetição das experiências realizadas neste trabalho, este anexo tem por objetivo apresentar os passos necessários para a compilação e execução da aplicação experimental desenvolvida. Antes de seguir os passos mostrados aqui é necessária a completa instalação do ACE/TAO conforme mostrado no Anexo A.

Antes de iniciar o processo de instalação (compilação) é necessário que se possua os seguintes arquivos:

Arquivo	Descrição
libcontrol.idl	Definição IDL das interfaces que compõem o sistema de teste e o mecanismo de adaptação.
offline.h	Informações sobre as tarefas que formam os sistema e que serão cadastradas junto ao serviço de escalonamento na fase *offline* do processo.
offline.cpp	Implementação dos procedimentos necessários para a fase *offline* do escalonamento. Com a execução da fase *offline* é gerado um arquivo denominado "*schedule.h*" que deve ser utilizado pela fase *runtime*.
schedule.h	É gerado na fase *offline* e contém as informações das tarefas cadastradas em "*offline.h*" e as informações completadas pelo serviço de escalonamento, como *handle* e prioridade para cada tarefa.
libcontrol_i.h	Definições das classes e métodos que foram especificados em IDL no "*libcontrol.idl*". Parte deste arquivo é gerado automaticamente pelo compilador IDL do TAO (gerado com o nome de "*libcontrolI.h*"), porém, foram feitas modificações necessárias para a implementação do mecanismo de adaptação. Não deve ser utilizada a versão gerada pelo compilador.
libcontrol_i.cpp	Implementação das classes definidas em "*libcontrol_i.h*". Parte do arquivo também é gerado pelo compilador IDL do TAO, porém, as modificações feitas em "*libcontrol_i.h*" também são refletidas neste arquivo, portanto, não se deve utilizar o arquivo gerado pelo compilador IDL diretamente.
buffer.h	Definição de uma classe auxiliar para armazenar as informações da execução de cada experiência e depois salvar em arquivo formato CSV (*Comma Separated Values*).
buffer.cpp	Implementação da classe *Buffer* definida em "*buffer.h*".
runtime.cpp	Implementação dos procedimentos necessários para uma execução da experiência. Cria e ativa as tarefas que são informadas como parâmetro para este programa. A forma de executar o *runtime* será detalhada mais adiante neste anexo.

De posse dos arquivos listados, serão mostrados aqui os passos necessários para a compilação da aplicação utilizando o *Microsoft Visual C++ 6.0* no sistema operacional *Microsoft Windows 2000 Advanced Server*. Outras configurações de compilador e sistema operacional, ainda que possíveis, não serão abordadas.

1. Criação do "workspace"

A aplicação é composta de três projetos e todos eles devem (porém não obrigatoriamente) estar em um mesmo *workspace*. Vai ser assumido que o *workspace* será criado no *drive* C com o nome de "control".

- Abra o Visual C e feche qualquer projeto ou *workspace* que possa por ventura estar sendo utilizado;
- Clique em "*File*" e "*New...*";

- Escolha a aba *"Workspaces"* e selecione o item *"Blank Workspace"*;
- No campo *"Workspace name:"* digite *"control"* (sem as aspas);
- No campo *"Location"* digite *"C:\control"*;
- Clique em OK e o *workspace* terá sido criado, salve-o.

2. Criação do projeto "libcontrol" e geração da biblioteca "libcontrol.lib"

Este projeto é responsável por gerar uma biblioteca com as classes *stub* e *skeleton* necessárias para a implementação de servidores que implementam as interfaces do arquivo *libcontrol.idl* ou para os clientes que acessam esses servidores.

- Clique com o botão direito do mouse no nome do projeto criado no passo anterior e escolha a opção *"Add New Project to Workspace..."*;
- Na aba *"Projects"* selecione o item *"Win32 Static Library"*;
- No campo *"Project Name"* digite *"libcontrol"*;
- No campo *"Location"* deixe apenas *"C:\control"*;
- Clique em OK, na próxima tela deixe os dois campos desmarcados e clique em *"Finish"*.
- Na tela do *workspace* clique com o botão direito do mouse no nome do projeto recém criado e escolha a opção *"Settings"*;
- Na aba *"C/C++"*, no campo *"Category:"* escolha o item *"Code Generation"*;
- No campo *"Use run-time library"* escolha a opção *"Debug Multithreaded DLL"*;
- Selecione a aba *"Library"* e no campo *"Output file name"* deixe apenas *"libcontrol.lib"*;
- Cique em OK;
- Acrescente o arquivo *"libcontrol.idl"* ao projeto clicando com o botão direito do mouse no item *"Source Files"* (logo abaixo do *"libcontrol files"*), selecionando a opção *"Add Files do Folder..."* e procurando o arquivo;
- Após acrescentado o arquivo *"libcontrol.idl"* é necessário informar ao Visual C como tratar com esse arquivo. Clique com o botão direito do mouse no nome do arquivo e selecione a opção *"Settings..."*;
- No campo *"Description"* insira:

```
Invoking TAO_IDL compiler on $(InputPath)
```

- No campo *"Comands"* insira:

```
$(ACE_ROOT)\bin\tao_idl.exe -Ge 1 -GI $(InputPath)
```

- No campo *"Outputs:"* insira:

```
$(InputName)C.cpp
$(InputName)C.h
$(InputName)C.i
$(InputName)S.cpp
$(InputName)S.h
$(InputName)S.i
$(InputName)S_T.cpp
$(InputName)S_T.h
$(InputName)S_T.i
```

- Clique em OK.

Após esses passos de configuração salve e compile o projeto. Serão gerados 11 arquivos (.h, .cpp e .i). Devem ser acrescentados na pasta *"Source Files"* os arquivos

"libcontrolC.cpp" e *"libcontrolS.cpp"*, e na pasta *"Header Files"* os arquvios *"libcontrolC.h"* e *"libcontrolS.h"*. Para acrescentar esses arquivos proceda da mesma forma como foi feito para acrescentar o arquivo *"libcontrol.idl"*. Finalmente, compile o projeto novamente e a biblioteca *"libcontrol.lib"* será gerada.

3. Criação e execução do projeto "offline"

O projeto *"offline"* é responsável pela implementação dos procedimentos necessários para cadastrar as tarefas da aplicação do serviço de escalonamento do TAO. Nesta fase do escalonamento são atribuídas as prioridades às tarefas e é gerado o arquivo *"schedule.h"* que deve ser utilizado no projeto *"runtime"*.

Passos para a criação do projeto:

- Clique com o botão direito do mouse no nome do projeto criado no passo anterior e escolha a opção *"Add New Project to Workspace..."*;
- Na aba *"Projects"* selecione o item *"Win32 Console Application"*;
- No campo *"Project Name"* digite *"offline"*;
- No campo *"Location"* deixe apenas *"C:\control"*;
- Clique em OK, na próxima tela deixe a opção *"An empty project"* selecionada e clique em *"Finish"*.
- Na tela do *workspace* clique com o botão direito do mouse no nome do projeto recém criado e escolha a opção *"Settings"*;
- Na aba *"Debug"*, no campo *"Executable for debug session"* deixe apenas *"offline.exe"*;
- Na aba *"C/C++"*, no campo *"Category:"* escolha o item *"Code Generation"*;
- No campo *"Use run-time library"* escolha a opção *"Debug Multithreaded DLL"*;
- Selecione a aba *"Link"* e no campo *"Output file name"* deixe apenas *"offline.exe"*;
- No campo *"Object/library modules"* insira as seguintes bibliotecas:

```
aced.lib taod.lib tao_portableserverd.lib
tao_cosnamingd.lib tao_rtschedd.lib
```

- Cique em OK;
- Acrescente o arquivo *"offline.cpp"* ao projeto clicando com o botão direito do mouse no item *"Source Files"* (logo abaixo do *"offline files"*), selecionando a opção *"Add Files do Folder..."* e procurando o arquivo. Acrescente também o arquivo *"offline.h"* na pasta *"Header Files"* de forma semelhante.

Após esses passos pode-se compilar o projeto e a próxima fase é a execução do mesmo. Para executa-lo siga os seguintes passos:

- Inicie o serviço de nomes do TAO (em rede local e com *multicast* habilitado):

```
$TAO_ROOT\orbsvcs\Naming_Service\Naming_Service.exe -m 1
```

- Inicie o serviço de eventos do TAO (em rede local):

```
$TAO_ROOT\orbsvcs\Event_Service\Event_Service.exe
```

- Execute o arquivo "offline.exe":

```
C:\control\offline.exe
```

Após a execução será criado o arquivo *"schedule.h"*.

4. Criação e compilação do projeto "runtime"

É o projeto responsável pela execução da aplicação experimental. Os passos para a sua criação são:

- Clique com o botão direito do mouse no nome do projeto criado no passo anterior e escolha a opção *"Add New Project to Workspace..."*;
- Na aba *"Projects"* selecione o item *"Win32 Console Application"*;
- No campo *"Project Name"* digite *"runtime"*;
- No campo *"Location"* deixe apenas *"C:\control"*;
- Clique em OK, na próxima tela deixe a opção *"An empty project"* selecionada e clique em *"Finish"*.
- Na tela do *workspace* clique no menu *"Project"* e na opção *"Dependencies"*;
- No campo *"Select property to modify"*, selecione o projeto *"runtime"*;
- Na caixa *"Dependent of the following project(s)"* marque o projeto *"libcontrol"* e clique em *"Close"*;
- De volta à tela do *workspace* clique com o botão direito do mouse no nome do projeto recém criado e escolha a opção *"Settings"*;
- Na aba *"Debug"*, no campo *"Executable for debug session"* deixe apenas *"runtime.exe"*;
- Na aba *"C/C++"*, no campo *"Category:"* escolha o item *"Code Generation"*;
- No campo *"Use run-time library"* escolha a opção *"Debug Multithreaded DLL"*;
- Selecione a aba *"Link"* e no campo *"Output file name"* deixe apenas *"runtime.exe"*;
- No campo *"Object/library modules"* insira as seguintes bibliotecas:

```
aced.lib taod.lib tao_rteventd.lib tao_portableserverd.lib
tao_cosnamingd.lib tao_rtcorbad.lib
tao_rtschedd.lib tao_svc_utilsd.lib
```

- Cique em OK;
- Acrescente os arquivos *"runtime.cpp"*, *"libcontrol_i.cpp"* e *"buffer.cpp"* ao projeto clicando com o botão direito do mouse no item *"Source Files"* (logo abaixo do *"runtime files"*), selecionando a opção *"Add Files do Folder..."* e procurando os arquivos. Acrescente também os arquivos *"schedule.h"*, *"libcontrol_i.h" e "buffer.h"* na pasta *"Header Files"* de forma semelhante.

Depois de executar os passos acima compile o projeto.

5. Execução da aplicação experimental

Pode ser utilizada apenas uma máquina para testes preliminares, porém os experimentos realizados neste trabalho utilizaram duas máquinas *Windows 2000 Advanced Server* em rede local.

Os passos necessários para se executar a aplicação são:

- Inicie o serviço de nomes do TAO (em rede local e com *multicast* habilitado):

```
$TAO_ROOT\orbsvcs\Naming_Service\Naming_Service.exe -m 1
```

- Inicie o serviço de eventos do TAO (em rede local):

```
$TAO_ROOT\orbsvcs\Event_Service\Event_Service.exe
```

- Execute o arquivo *"offline.exe"*:

```
C:\control\offline.exe
```

- Execute o arquivo *"runtime.exe"* nas duas máquinas simultaneamente com as seguintes linhas de comando (uma para cada máquina):

```
C:\control\runtime.exe sensorA atuadorA
                       atuadorB alarme1 operador1
C:\control\runtime.exe sensorB alarme2 operador2
                       hsitorico1 historico2 AdaptiveService
```

A execução da aplicação, caso ocorra sem problemas, gerará 15 arquivos com extensão CSV (*Comma Separated Values*) que podem ser importados para o *Microsoft Excel*. Os arquivos gerados têm como nome uma concatenação entre o nome da tarefa e as letras "be" (que significa *"before event"*, nestas planilhas pode-se observar os tempos de chegada de eventos e a média do período das tarefas) e "ae" (que significa *"after event"* e onde estão os tempos de resposta das tarefas e média dos atrasos). No caso do serviço de adaptação, também é gerada uma planilha com as letras "dyt", onde estão os valores de DY(t) em cada evento e também o valor do fator de adaptação.

Bibliografia

[ABD 97] ABDELZAHER, T. F.; ATKINS, E. M.; SHIN, K. G. QoS negotiation in real-time systems and its application to automatic flight control. In: IEEE REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, RTAS, 3., 1997. **Proceedings…** Montreal: IEEE Computer Society Press, 1997.

[ABD 98] ABDELZAHER, E. M.; SHIN, K. G. End-host Architecture for QoS-Adaptive Communication. In: IEEE REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, RTAS, 4., 1998. **Proceedings…** Denver: IEEE Computer Society Press, 1998.

[AUD 93] AUDSLEY, N.C. et al. Applying New Scheduling Theory to Static Priority Preemptive Scheduling. **Software Engineering Journal**, [S.l.], v. 8, n. 5, p. 284-292, 1993.

[BEC 99] BECCARI, G.; CASELLI, S.; REGGIANI, M.; ZANICHELLI, F. Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 11., 1999. **Proceedings…** York: [s.n.], 1999.

[BRA 98] BRANDT, S.; NUTT, G.; BERK, T.; MANKOVICH, J. A dynamic quality of service middleware agent for mediating application resource usage. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM, RTSS, 19., 1998. **Proceedings…** Madri: IEEE Computer Society Press, 1998.

[BRO 2000] BROSE, G. **JacORB:** Implementation and Design of a Java ORB. Disponível em: <http://jacorb.inf.fu-berlin.de>. Acesso em: jul. 2000.

[DAB 2000] DABKE, P. Enterprise Integration via CORBA-based information agents. **IEEE Internet Computing**, [S.l.], v. 3, p. 49-57, Sept./Oct. 1999.

[EID 97] EIDE, E. et al. Flick: A Flexible, Optimizing IDL Compiler. **SIGPLAN Notices**, New York, v. 32, n. 5, p. 44-56, May 1997. Trabalho apresentado na ACM SIGPLAN Conference on Programming Language Design and Implementation, SIGPLAN, 1997.

[FAR 2000] FARINES, Jean-Marie; FRAGA, Jonidas S.; OLIVEIRA, Rômulo. **Sistemas de Tempo Real.** São Paulo: IME-USP, 2000.

[FEL 2000] FELBER, Pascal; GUERRAOUI, Rachid. Programming with Object Groups in CORBA. **IEEE Concurrency**, [S.l.], v. 8, n. 1, p. 48-58, Jan./Mar. 2000.

[FEN 2000] FENG, W.; SYYID, U.; LIU, J. W.-S. **Providing for an Open, Real-Time CORBA.** Disponível em: <http://citeseer.nj.nec.com/feng97providing.html>. Acesso em: dez. 2000.

[GEC 97] GECSEI, J. Adaptation in Distributed Multimedia Systems. **IEEE Multimedia**, [S.l.], v. 4, n. 2, p. 58-66, Apr./June 1997.

[GIL 2000] GILL, Christopher D.; LEVINE, David L.; SCHMIDT, Douglas C. **The Design and Performance of a Real-Time CORBA Scheduling Service.** Disponível em: <http://citeseer.nj.nec.com/gill99design.html>. Acesso em: dez. 2000.

[GOE 99] GOEL, Ashvin; STEERE, David; PU, Calton; WALPOLE, Jonathan. **Adaptive Resource Management Via Modular Feedback Control.**

Technical Report 99-03, Oregon Graduate Institute, Computer Science and Engineering, Jan. 1999. Disponível em: <http://citeseer.nj.nec.com/goel99adaptive.html>. Acesso em: nov. 2001.

[HAM 97] HAMDAOUI, M.; RAMANATHAN, P. Evaluating Dynamic Failure Probability for Streams with (m,k)-Firm Deadlines. **IEEE Transactions on Computers**, [S.l.], v. 46, n. 12, p. 1325-1337, Dec. 1997.

[HAR 97] HARRISON, Timothy H.; LEVINE, David L.; SCHMIDT, Douglas C. The Design and Performance of a Real-Time CORBA Event Service. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, OOPSLA, 1997. Disponível em: <http://citeseer.nj.nec.com/harrison97design.html>. Acesso em: ago. 2001.

[HEN 99] HENNING, Michi; VINOSKI, Steve. **Advanced CORBA Programming with C++.** USA: Addison-Wesley, 1999.

[JEN 85] JENSEN, E. D.; LOCKE, C. D.; TOKUDA, H. A Time-Driven Scheduling Model for Real-Time Operating Systems. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM, RTSS, 1985. **Proceedings…** [S.l.]: IEEE Computer Society Press, 1985.

[LIU 73] LIU, C.L.; LAYLAND, J.W. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. **Journal of the ACM**, [S.l.], v. 20, n. 1, p. 46-61, Jan. 1973.

[LIU 94] LIU, J.W.S.; SHIH, W.-K.; LIN, K.-J.; BETTATI, R.; CHUNG, J.-Y. Imprecise Computations. **Proceedings of the IEEE**, [S.l.], v. 82, n. 1, p. 83-94, Jan. 1994.

[LU 2000] LU, Chenyang; STANKOVIC, John A.; ABDELZAHER, Tarek F.; TAO, Gang; SON, Sang H.; MARLEY, Michael. Performance Specifications and Metrics for Adaptive Real-Time Systems. In: IEEE REAL-TIME SYSTEMS SYMPOSIUM, RTSS, 21., 2000. **Proceedings…** Orlando: IEEE Computer Society Press, 2000.

[MCE 2000] MCELHONE, Charlie; BURNS, Alan. Scheduling Optional Computations for Adapttive Real-Time Systems. **Journal of Systems Architectures,** [S.l.], v. 46, n. 1, p. 49-77, Jan. 2000. Disponível em: <http://citeseer.nj.nec.com/mcelhone00scheduling.html>. Acesso em: nov. 2001.

[MON 99] MONTEZ, Carlos B. **Um modelo de programação e uma abordagem de escalonamento adaptativo usando RT-CORBA.** 1999. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis.

[MOW 97] MOWBRAY, Thomas J.; MALVEAU, Raphael C. **CORBA Design Patterns.** USA: Wiley Computer Publishing, 1997.

[MOW 98] MOWBRAY, Thomas J.; RUH, William A. **Inside CORBA:** Distributed Object Standards and Applications. USA: Addison-Wesley, 1998.

[NET 98] NETT, E.; GERGELEIT, M.; MOCK, M. An Adaptive Approach to Object-Oriented Real-Time Computing. In: INTERNATIONAL SYMPOSIUM ON OBJECT-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, ISORC, 2., 2000. **Proceedings...**

Kyoto: IEEE Computer Society Press, 1998.

[OLI 97] OLIVEIRA, Rômulo S. **Escalonamento de Tarefas Imprecisas em Ambiente Distribuído.** 1997. Tese (Doutorado em Engenharia Elétrica) – Programa de Pós-graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis.

[OLI 98] OLIVEIRA, Rômulo S. Mecanismos de Adaptação para Aplicações Tempo Real na Internet. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, SEMISH, 1998. **Anais...** Belo Horizonte: Sociedade Brasileira de Computação, 1998.

[OLI 99] OLIVEIRA, Rômulo S.; FURTADO, Olinto J.V. Um Mecanismo de Adaptação para Aplicações Tempo Real Baseado em Computação Imprecisa e Reflexão Computacional. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 13., 1999. **Anais...** Florianópolis: Sociedade Brasileira de Computação, 1999.

[OMG 2000] OMG. **What is the OMG?** Disponível em: <http://www.omg.org/memberservices/index.htm#whatisOMG>. Acesso em: ago. 2000.

[OMG 2000a] OMG. **Search for Member Companies.** Disponível em: <http://cgi.omg.org/cgi-bin/membersearch.pl>. Acesso em: ago. 2000.

[OMG 2000b] OMG. **Introduction to OMG's Specifications.** Disponível em: <http://www.omg.org/gettingstarted/specintro.htm>. Acesso em: ago. 2000.

[OMG 2000c] OMG. **Realtime CORBA.** Alcatel; Hewlett-Packard Company; Lucent Technologies, Inc.; Object-Oriented Concepts, Inc.; Sun Microsystems, Inc.; Tri-Pacif. OMG Document orbos/98-01-08. Disponível em: <http://www.cs.wustl.edu/~schmidt/PDF/RT-ORB-std.pdf>. Acesso em: out. 2000.

[OMG 2000d] OMG. **Realtime CORBA:** Joint Revised Submission. Alcatel; Hewlett-Packard Company; Highlander Communication, L.C.; INPRISE Corporation; IONA Technologies; Lockheed Martin Federal Systems, Inc.; Lucent Technologies, Inc.; Nortel Networks; Objective Interface Systems, Inc.; Object-Oriented Concepts, Inc.; Sun Microsystems, Inc.; Tri-Pacific Software, Inc. OMG TC Document orbos/98-10-05. Disponível em: <http://www.cs.wustl.edu/~schmidt/PDF/RT-ORB-std-new.pdf >. Acesso em: out. 2000.

[OMG 2000e] OMG. **Realtime CORBA:** Joint Revised Submission. Alcatel; Hewlett-Packard Company; Highlander Communication, L.C.; INPRISE Corporation; IONA Technologies; Lockheed Martin Federal Systems, Inc.; Lucent Technologies, Inc.; Nortel Networks; Objective Interface Systems, Inc.; Object-Oriented Concepts, Inc.; Sun Microsystems, Inc.; Tri-Pacific Software, Inc. OMG TC Document orbos/99-02-12. Disponível em: <http://www.omg.org/cgi-bin/doc?orbos/99-02-12 >. Acesso em: out. 2000.

[ORF 98] ORFALI, Robert; HARKEY, Dan. **Client/Server Programming with JAVA and CORBA.** 2 nd ed. USA: Wiley Computer Publishing, 1998.

[ORY 2000] O'RYAN, Carlos; SCHMIDT, Douglas C. et al. **Evaluating Policies and Mechanisms to Support Distributed Real-Time Applications with CORBA.** Disponível em: <http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html>. Acesso em: nov. 2000.

[POP 98] POPE, Alan. **The CORBA Reference Guide:** Understanding the Common Object Request Broker Architecture. USA: Addison-Wesley, 1998.

[SCH 2000] SCHMIDT, Douglas C.; LEVINE, David L.; CLEELAND, Chris. **Architectures and Patterns for Developing High-performance, Real-time ORB Endsystems.** 1998. Disponível em: <http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html>. Acesso em: out. 2000.

[SCH 2000a] SCHIMIDT, Douglas C.; LEVINE, David L. **Developing Standard Real-time CORBA Applications using TAO Object Request Broker – A proposal for a TAO Real-time CORBA Developers Guide.** Disponível em: <http://www.cs.wustl.edu/~schmidt/PDF/RT-CORBA-proposal.pdf >. Acesso em: out. 2000.

[SCH 2000b] SCHIMIDT, Douglas C.; KUHNS, Fred. An Overview of the Real-Time CORBA Specification**. Computer**, [S.l.], v. 33, n. 6, p. 56-63, 2000.

[SCH 2000c] SCHMIDT, Douglas C.; LEVINE, David L.; MUNGEE, Sumedh. The Design of the TAO Real-Time Object Request Broker. **Computer Communications Journal**, [S.l.], 1997. Disponível em: <http://citeseer.nj.nec.com/138439.html>. Acesso em: dez 2000.

[SCH 2000d] SCHMIDT, Douglas C. et al. **TAO:** a High-performance ORB Endsystem Architecture for Real-time CORBA. RFI in response to the OMG Special Interest Group on Real-time CORBA. Disponível em: <http://citeseer.nj.nec.com/schmidt97tao.html>. Acesso em: dez 2000.

[SCH 2000e] SCHMIDT, Douglas C. et al. An ORB Endsystem Architecture for Statically Scheduled Real-time Applications. In: IEEE WORKSHOP ON MIDDLEWARE FOR REAL-TIME SYSTEMS AND SERVICES, 1997. Disponível em: <http://citeseer.nj.nec.com/schmidt97orb.html>. Acesso em: dez. 2000.

[SHI 99] SHIN, K.G.; MEISSNER, C. L. Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment. In: EUROMICRO CONFERENCE ON REAL-TIME SYSTEMS, 11., 1999. **Proceedings…** York: [s.n.], 1999.

[SHO 2000] SHOKRI, Eltefaat; SHEU, Phillip. Guest Editor's Introduction: Real-Time Distributed Object Computing – An Emerging Field. **Computer**, [S.l.], v. 33, n. 6, p. 45-46, 2000.

[SYY 2000] SYYID, Umar. **The Adaptive Communication Environment:** "ACE". A Tutorial. Hughes Network Systems. Disponível em: <http://www.cs.wustl.edu/~schmidt/PDF/ACE-tutorial.pdf >. Acesso em: dez. 2000.

[TAO 2000] TAO: The ACE ORB. Disponível em: <http://siesta.cs.wustl.edu/~schmidt/TAO.html>. Acesso em: nov. 2000.

[VER 2000] VERTEL Corporation. Disponível em: <http://www.vertel.com>. Acesso em: nov. 2000.

[WEL 98] WELCH, L. R.; SHIRAZI, B. A.; RAVINDRAN, B. Adaptive Resource Management for Scalable, Dependable Real-Time Systems: Middleware Services and Applications to Shipboard Computing Systems. In: IEEE REAL-TIME TECHNOLOGY AND APPLICATIONS SYMPOSIUM, RTAS, 1998. **Proceedings…**

Denver: IEEE Computer Society Press, 1998.