UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Camada Groupware: Uma Camada de Suporte a Aplicações Cooperativas em Ambientes Distribuídos Abertos

Este exemplar corresponde à redação final da tese defendida por FERNANDO CORTEZ SICA e aprovada pela Comissão julgadora em 20/12/96.
Waldomiro Loyolla Orientador

autor: **Fernando Cortez Sica**
Orientador: **Prof. Dr. Waldomiro P.D.C. Loyolla**

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica, Área de concentração: Automação Industrial

Campinas - SP Brasil
1996

UNICAMP
BIBLIOTECA CENTRAL

UNIDADE BC
N.º CHAMADA:
T/UNICAMP
Si12c
V. Ex.
TOMBO BC/ 30383
PROC. 281/97
C D
PREÇO R$ 11,00
DATA 23/05/97
Nº CPD

CM-00099317-2

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si12c

Sica, Fernando Cortez
Camada groupware: uma camada de suporte a aplicações cooperativas em ambientes distribuidos abertos / Fernando Cortez Sica.--Campinas, SP: [s.n.], 1996.

Orientador: Waldomiro P. D. C. Loyolla.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas abertos (Computadores). 2. Processamento distribuido. 3. Redes de computadores. 4. Sistemas operacionais distribuidos (Computadores). I. Loyolla, Waldomiro P. D. C.. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Índice

Resumo x
Abstract xi
Capítulo 1- Introdução 1
Capítulo 2 - Serviço Cooperativo Suportado por Computador 4
2.1 - Introdução 4
2.2 - Caracterização dos Sistemas CSCW 5
2.2.1 - Aspectos Humanos do Trabalho em Grupo 6
2.2.1.1 - Aspecto Individual Humano 6
2.2.1.2 - Aspecto Organizacional 6
2.2.1.3 - Ponto de Vista da Dinâmica de Grupo 7
2.2.2 - Aspectos Tecnológicos 8
2.2.2.1 - Suporte às Atividades de Grupo 8
2.2.2.2 - Suporte ao Espaço de Trabalho Compartilhado 8
2.2.2.3 - Suporte ao Compartilhamento de Informação 9
2.2.2.4 - Suporte à Comunicação 9
2.2.2.5 - Suporte à Transparência 9
2.2.3 - Classificação Espaço/Tempo 9
2.3 - Cenários Exemplos de Aplicações CSCW 13

Capítulo 3 - Processamento Distribuído Aberto (ODP) 17
3.1 - Introdução 17
3.2 - Padronização RM-ODP 17
3.2.1 - Pontos de Vista do Modelo de Referência ODP 18
3.2.1.1 - Ponto de Vista Empresarial 18
3.2.1.2 - Ponto de Vista de Informação 19
3.2.1.3 - Ponto de Vista Computacional 20
3.2.1.4 - Ponto de Vista de Engenharia 20
3.2.1.5 - Ponto de Vista Tecnológico 20
3.2.2 - Aspectos Gerais de Modelagem 21
3.2.2.1 - Conceitos de Modelagem Básica 21
3.2.2.1.1 - Conceito de Objetos 21
3.2.2.1.2 - Encapsulamento e Abstração 21
3.2.2.1.3 - Comportamento e Estado 22
3.2.2.1.4 - Interface 22
3.2.2.1.5 - Modelagem de Objetos em ODP 22
3.2.2.2 - Conceitos de Especificação 22
3.2.2.2.1 - Composição e Decomposição 23
3.2.2.2.2 - Templates 23
3.2.2.2.3 - Papéis 24
3.2.2.2.4 - Tipos e Classes 24
3.2.2.2.5 - Hierarquia Classe/Sub-Classe 24
3.2.2.2.6 - Hierarquia Classe Base/Classe Derivada 25
3.2.2.3 - Conceitos Arquiteturais 25

3.2.2.3.1 - Nome 25
3.2.2.3.2 - Grupos 25
3.2.2.3.3 - Domínios 25
3.2.2.3.4 - Contrato 26
3.2.2.3.5 - *Liaison* e *Binding* 26
3.2.2.3.5.1 - Binding Explícito 26
3.2.2.3.5.2 - Binding Implícito 27
3.3 - Arquitetura CORBA 28
3.3.1 - Estrutura do Object Requester Broker 29
3.3.1.1 - Núcleo ORB 32
3.3.1.2 - Implementações de Objetos 32
3.3.1.3 - Referências de Objetos 32
3.3.1.4 - Linguagem de Definição de Interfaces (IDL) 32
3.3.1.5 - Interface de Invocação Dinâmica 32
3.3.1.6 - *Stubs* 33
3.3.1.7 - *Skeleton* Estático 33
3.3.1.8 - *Skeleton* Dinâmico 33
3.3.1.9 - Adapatdores de Objetos 33
3.3.1.10 - Interface ORB 33
3.3.1.11 - Repositório de Interface 33
3.3.1.12 - Repositório de Implementação 34
3.4 - Plataforma *Orbix* 34
3.4.1 - Conceitos *Orbix* 34
3.4.1.1 - Proxy 34
3.4.1.2 - Modos de Ativação 35
3.4.1.3 - *Threads* 35
3.4.1.4 - Implementação de Interfaces 36
3.4.1.4.1 - *BOAImpl Approach* 36
3.4.1.4.2 - *TIE Approach* 36

Capítulo 4 - Técnica de Modelagem de Objetos (OMT) 37
4.1 - Introdução 37
4.2 - Conceitos Básicos 37
4.2.1 - Identidade 37
4.2.2 - Classe 38
4.2.3 - Polimorfismo 38
4.2.4 - Herança 38
4.2.5 - Abstração 39
4.2.6 - Encapsulamento 39
4.3 - Modelagem Estática 39
4.3.1 - Classes e Objetos 39
4.3.2 - Atributos 39
4.3.3 - Operações e Métodos 40
4.3.4 - *Links* e Associações 41
4.3.5 - Cardinalidade 41
4.3.6 - Atributos de Associação 42
4.3.7 - Denominação de Papéis 42
4.3.8 - Agregação 42

4.3.9 - Generalização/Especialização 43
4.4 - Modelagem Dinâmica 44
4.4.1 - Diagrama de Estados 44
4.4.2 - Histórico de Eventos 45
4.4.3 - Cenários 46
4.5 - Modelagem Funcional 46

Capítulo 5 - Especificação da Camada Groupware 48
5.1 - Introdução 48
5.2 - Plataforma Multiware 48
5.2.1 - Camada de Baixo Nível 49
5.2.2 - Camada Middleware 49
5.2.2.1 - Sub-Camada de Processamento Multimídia 49
5.2.2.2 - Sub-Camada ODP 50
5.2.3 - Camada Groupware 51
5.3 - Serviços Oferecidos pela Camada Groupware 51
5.3.1 - Serviços de Contactação 52
5.3.1.1 - Serviços de Criação, Encerramento e Ativação de Grupos Cooperativos 52
5.3.1.2 - Serviços de Adição de Novos Membros a Grupos já em Andamento 52
5.3.1.3 - Serviços de Ativação das Ações Cooperativas Locais 53
5.3.1.4 - Serviços de Informação sobre Grupos 53
5.3.2 - Serviços de Manutenção 53
5.3.2.1 - Serviços de Manipulação dos Canais de Informação 53
5.3.2.2 - Serviços de Envio de Mensagens 54
5.3.2.3 - Serviços de Questionamento (votação) 54
5.3.2.4 - Serviços de Saída/Interrupção do Membro Frente ao Grupo 54

Capítulo 6 - Projeto da Camada Groupware 55
6.1 - Introdução 55
6.2 - Sub-Camadas da Camada Groupware 55
6.3 - Localização das Referências dos Objetos da Camada Groupware 56
6.4 - Objetos Integrantes da Camada Groupware 56
6.4.1 - Objeto ContactServiceObject 58
6.4.2 - Objeto ServiceFactoryObject 59
6.4.3 - Objeto CSCWServicesObject 60
6.4.4 - Objeto ViewSharerObject 61
6.4.5 - Objeto MembersManagerObject 61
6.4.6 - Objeto MembersParticipantObject 62
6.4.7 - Objeto ResourcesManagerObject 63
6.4.8 - Objeto ResourcesObject 64
6.5 - Exemplo de Utilização 65

7 - Aspectos de Implementação da Camada Groupware..... 67
7.1 - Introdução..... 67
7.2 - Atributos dos Objetos da Camada Groupware..... 67
7.2.1 - Constantes Presentes na Camada Groupware..... 67
7.2.2 - Tipos Enumerados da Camada Groupware..... 67
7.2.3 - Estruturas Presentes na Camada Groupware..... 68
7.2.4 - Atributos do Objeto ContactServiceObject..... 69
7.2.5 - Atributos do Objeto ServiceFactoryObject..... 69
7.2.6 - Atributos do Objeto CSCWServiceObject..... 69
7.2.7 - Atributos do Objeto MembersManagerObject..... 70
7.2.8 - Atributos do Objeto MemberParticipant..... 71
7.2.9 - Atributos do Objeto ResourcesManagerObject..... 71
7.2.10 - Atributos do Objeto ResourcesObject..... 72
7.3 - Métodos Implementados..... 72
7.3.1 - ContactServiceObject..... 73
7.3.1.1 - ConstructNewService..... 73
7.3.1.2 - CloseGroup..... 74
7.3.1.3 - SearchGroup..... 75
7.3.1.4 - ActiveGroup..... 76
7.3.1.5 - InitMember..... 77
7.3.1.6 - AddNewMembe..... 77
7.3.1.7 - GetCreatedInfo..... 78
7.3.2 - ServiceFactoryObject..... 79
7.3.2.1 - ConstructNewService..... 79
7.3.3 - CSCWServiceObject..... 81
7.3.3.1 - SetStatus..... 81
7.3.3.2 - GetStatus..... 81
7.3.3.3 - ActiveGroup..... 82
7.3.3.4 - ActiveIGroup/ActiveAGroup/ActiveVGroup..... 82
7.3.3.5 - ConnectMembersManager..... 83
7.3.3.6 - InitMember..... 84
7.3.3.7 - AddNewMember..... 85
7.3.3.9 - CloseGroup..... 85
7.3.3.9 - SetStartTime..... 87
7.3.3.10 - SetDurationTime..... 87
7.3.3.11 - AppendViewSharer..... 87
7.3.4 - MembersManagerObject..... 88
7.3.4.1 - ConnectMembers..... 88
7.3.4.2 - ConnectMember..... 89
7.3.4.3 - InitXMember..... 91
7.3.4.4 - ActiveIGroup/ActiveAGroup/ActiveVGroup..... 92
7.3.4.5 - CloseGroup..... 93
7.3.4.6 - GetInfoMember..... 93
7.3.4.7 - AnswerQuestion..... 94
7.3.4.8 - SendGeneralMessage..... 94
7.3.4.9 - SendFormalMessage..... 95
7.3.4.10 - ChangeCurrentMember..... 95
7.3.4.11 - InterruptMember..... 96

7.3.5 - MemberParticipant 97
7.3.5.1 - AppendToResourcesManager 97
7.3.5.2 - AppendToNewResourcesManager 98
7.3.5.3 - SetIMemMan/SetAMemMan/SetVMemMan 99
7.3.5.4 - GetStatus 100
7.3.5.5 - SetStatus 100
7.3.5.6 - OpenDInformalChannel 100
7.3.5.7 - OpenSInformalChannel 101
7.3.5.8 - CloseRDInformalChannel 102
7.3.5.9 - CloseInformalChannel 102
7.3.5.10 - Interrupt 103
7.3.5.11 - SendFormalMessage 104
7.3.5.12 - SendInformalMessage 104
7.3.5.13 - SendGeneralMessage 105
7.3.5.14 - ShowFomalMessage 106
7.3.5.15 - ShowGeneralMessage 106
7.3.5.16 - GetMemberName 107
7.3.5.17 - CloseGroup 107
7.3.5.18 - AnswerQuestion 108
7.3.5.19 - InitCMember/InitNMember 108
7.3.5.20 - ChangeCurrentMember 109
7.3.6 - ResourcesManagerObject 110
7.3.6.1 - AppendResources 110
7.3.6.2 - OpenResources 111
7.3.6.3 - OpenInformalChannel 111
7.3.6.4 - CloseInformalChannel 112
7.3.6.5 - ShowFomalMessage 113
7.3.6.6 - ShowGeneralMessage 113
7.3.6.7 - SetLCMemPar/SetLNMemPar/ SetSCMemPar/SetSNMemPar 114
7.3.6.8 - ParcialCloseResources 114
7.3.6.9 - CloseLGroup/CloseSGroup 115
7.3.6.10 - ConnectMember 115
7.3.6.11 - SetProprietary 116
7.3.7 - ResourcesObject 116
7.3.7.1 - ShowMessage 116
7.3.7.2 - OpenDisplay 117
7.3.7.3 - Kill 118
7.4 - Objetos Servidores 118

Capítulo 8 - Exemplo de Utilização da Camada Groupware 120
8.1 - Introdução 120
8.2 - Manipulação da Plataforma Distribuída Orbix 120
8.3 - Descrição da Aplicação de Tele-Conferência 121
8.4 - Exemplificaçào dos Serviços Fornecidos pela Camada Groupware 123

Capítulo 9 - Conclusões e Trabalhos Futuros 129

Anexo A - Arquivos Pertencentes à Camada Groupware..... 131

Anexo B - Exemplo do Makefile..... 133

Anexo C - Exemplo de uma Aplicação Cooperativa: Tele-Conferência..... 136

Anexo D - Exemplo de Registro no Repositório de Interfaces..... 154

Referências Bibliográficas..... 155

Índice de Figuras

2.1 - Relação Espaço/Tempo 10
2.2 - Divisão das classes de serviços CSCW por localização temporal.. 10
2.3 - Divisão das classes de serviços CSCW
por localização geográfica 11
3.1 - Comparação entre Pontos de Vista (RM-ODP)
e Engenharia de Software 18
3.2 - Sistema Cliente-Servidor 18
3.3 - Arquitetura de um Objeto 21
3.4 - Decomposição de Objetos 23
3.5 - Composição de Objetos 23
3.6 - Instanciação de Objetos a partir de *templates* 24
3.7 - Papéis desempenhados pelos objetos nas interações 24
3.8 - Conexão de dois objetos (*A* e *B*) com binding explícito 26
3.9 - Conexão de dois objetos (*A* e *B*) com binding implícito 27
3.10 - Objeto cliente em seu estado inicial 27
3.11 - Instanciação de um objeto servidor 27
3.12 - Criação de interface *x:I1* 27
3.13 - Anúncio da existência de *x:I1* 28
3.14 - Criação da interface *y:I2* por parte de *A* 28
3.15 - Objeto *A* anuncia necessidade de 28
comunicação com alguma interface *I* 28
3.16 - Encontro da interface *x:I* por *A* 28
3.17 - *Liaison* envolvendo as interfaces *x:I1* e *y:I2* 28
3.18 - Interação do Orb entre cliente e servidor 30
3.19 - Aquitetura ORB 30
3.20 - Relação entre definições, implementações e repositórios 31
3.21 - Etapa de Criação *IDL C++* 34
3.22 - Relacionamento remoto de dois objetos por meio de um *proxy* ... 35
4.1 - Herança entre Classes Bases e Derivadas 38
4.2 - Representação inicial de classe e objetos (a) e (b) 39
4.3 - Representação de classe e objetos com atributos 40
4.4 - Exemplo de uma classe completa 40
4.5 - Associaçào entre as entidades *Universidade* e *Reitor* 41
4.6 - Associações *n-árias* entre classes 41
4.7 - Inserção de atributos no relacionamento
entre *Residência* e *Família* 42
4.8 - Papéis desempenhados pelas classes *Indústria* e *Pessoa* 42
4.9 - Agregação entre os componentes de um automóvel 43
4.10 - Representação de Associações Hierárquicas 43
4.11 - Etapas no processo de implementação de um programa 44
4.12 - Operações envolvidas durante o processo de compilação 45
4.13 - Processo de Compilação com e sem a
ocorrência de erro (a) e (b) 45
4.14 - Exemplo de Modelagem Funcional 46

5.1 - Estruturação da Plataforma Multiware 48
5.2 - Divisão Estrutural da sub-camada ODP 50
5.3 - Divisão Estrutural do sub-nível ODP-Support 50
5.4 - Visão dos Serviços Exportados pela Camada Frente aos Usuários 51
6.1 - Distribuição Funcional das Sub-Camadas Cooperativas 55
6.2 - Hierarquia dos Objetos da Camada Groupware 57
6.3 - Objetos Primários da Camada Groupware 58
6.4 - Conexão entre um membro externo e um serviço 58
6.5 - Ligações a partir do CSCWServicesObject 60
6.6 - Tipos especializados do CSCWServicesObject 61
6.7 - Especializações do objeto MembersManagerObject 62
6.8 - Especializações do MembersParticipantObject 63
6.9 - Serviço Cooperativo com recursos locais 63
6.10 - Serviço Cooperativo com recursos comuns ao grupo 64
6.11 - Especializações do objeto ResourcesManagerObject 64
6.12 - Objetos representativos dos recursos 65
6.13 - Exemplo da distribuição dos objetos em um grupo com recursos locais 66
7.1 - Criação de um objeto *ServiceFactoryObject* 73
7.2 - Fechamento de um serviço cooperativo 74
7.3 - Obtenção de informações básicas de um grupo criado 75
7.4 - Início da ação cooperativa 76
7.5 - Iniciação de um membro 77
7.6 - Processo para o pedido de adição a partir de membros externos 78
7.7 - Obtenção de informações sobre grupos criados 79
7.8 - Etapa de criação de um novo grupo a partir do *ContactServiceObject* 80
7.9 - Alteração do atributo de estado 81
7.10 - Captação do atributo de estado 81
7.11 - Processo de ativação de um grupo 82
7.12 - Etapa de anexação de um *MembersManagerObject* 83
7.13 - Iniciação de um membro 84
7.14 - Anexação de um novo membro 85
7.15 - Etapas para o fechamento de um grupo 86
7.16 - Configuração inicial de um membro participante 89
7.17 - Pedido de anexação gerando um sistema de consulta ao chairman 90
7.18 - Resposta do chairman não enviada ou negativa 90
7.19 - Resposta enviada a partir do chairman do grupo 90
7.20 - Configuração inicial de um membro participante 91
7.21 - Ativação de um membro 92
7.22 - Instanciação do atributo *CSCW* do *MembersManager* 92
7.23 - Fechamento de um grupo a nível do *MembersManager* 93
7.24 - Utilização do método *AnswerQuestion* 94
7.25 - Envio de uma mensagem formal a nível do *MemberParticipant* 95
7.26 - Mudança do membro corrente 96
7.27 - Serviço cooperativo com recursos compartilhados 97

7.28 - Anexação do membro a um *ResourcesManager* já instanciado... 97
7.29 - Serviços cooperativos com recursos locais... 98
7.30 - Anexação de um membro a um *ResourceManager* com recursos locais... 99
7.31 - Abertura de um canal informal a nível do membro destino... 101
7.32 - Abertura de um canal de comunicação por parte do membro fonte... 101
7.33 - Fechamento de um canal informal a níve do requisitante... 103
7.34 - Interrupção de um membro... 103
7.35 - Envio de uma mensagem formal... 104
7.36 - Envio de uma mensagem informal... 105
7.37 - Envio de uma mensagem genérica... 105
7.38 - Exibição de uma mensagem formal... 106
7.39 - Exibição de uma mensagem genérica... 106
7.40 - Fechamento do grupo a nível do *MemberParticipant*... 107
7.41 - Mecanismo de resposta às questões pendentes... 108
7.42 - Etapa de iniciação de um membro frente ao *ResourcesManager* 109
7.43 - Requisição da mudança de membro corrente... 109
7.44 - Adição de recursos durante a construção do grupo cooperativo. 110
7.45 - Abertura dos recursos para a iniciação do cooperativismo... 111
7.46 - Abertura de um canal informal de comunicação... 112
7.47 - Fechamento de um canal informal a nível do *ResourcesManager*... 112
7.48 - Exibição de mensagem formal a nível do *ResourcesManager*... 113
7.49 - Exibição de mensagem geral a nível do *ResourcesManager*... 113
7.50 - Fechamento parcial dos recursos associados ao *ResourcesManager*... 114
7.51 - Fechamento de um grupo cooperativo... 115
7.52 - Exibição de uma mensagem... 117
7.53 - Abertura de uma janela... 117
7.54 - Fechamento de uma janela... 118
7.55 - Estado inicial do Ambiente *A*... 119
7.56 - Estado do Ambiente *A* após o servidor receber os pedidos para a criação... 119
7.57 - Objeto Servidor atendendo ao pedido de fechamento do grupo.. 119
7.58 - Estado final de *A* após a série de requisições... 119
8.1 - Relacionamento entre a aplicação, a Camada Groupware e o Orbix... 120
8.2 - Interação entre objetos cliente e servidor através do daemon Orbix... 121

Resumo

A expansão da conectividade faz com que apareçam novas filosofias e frentes de pesquisa dentro do ramo das redes de computadores e computação distribuída. Uma destas frentes enfatiza o chamado *Trabalho Cooperativo Suportado por Computador* (CSCW), o qual tem o objetivo de englobar várias pessoas dentro de um universo cooperativo no compartilhamento de recursos.

O objetivo desta tese consiste em descrever aspectos de implementação da camada *Groupware* (Camada de Suporte para Aplicações Cooperativas) a qual fornece serviços e interfaces para o estabelecimento, manutenção e fechamento de aplicações cooperativas genéricas.

Abstract

The conectivity expansion is responsible by the appear of the new filosofies and research lines at computer networks and distributed computation fields. One of the research line hang the intituled *Computer Supported Cooperative Work* (CSCW), which objectifys to englobe many persons in the cooperative universe at resource sharely.

This thesis describe the *Groupware Layer* (Cooperative Aplications Support Layer) implementation aspects. This layer supply services to establish, manager and close generics cooperatives aplications.

Capítulo 1 - Introdução

Observa-se, atualmente, uma crescente evolução dos sistemas de comunicação, interligando os mais diversos ambientes computacionais, que podem variar desde máquinas com maior poderio de processamento até assistentes pessoais.

Unem-se ao escopo das redes de computadores de alto desempenho, as aplicações multimídia, as quais integram dados, texto, voz e imagens e que vêm despertando grande interesse tanto de pesquisadores quanto de empresas da área.

Tendo como um dos enfoques principais a conectividade, estabeleceu-se uma nova área de pesquisa, chamada de *Trabalho Cooperativo Suportado por Computador* (**CSCW** - *Computer Supported Cooperative Work*). Tal área tem como meta integrar o trabalho de várias pessoas em um objetivo comum, estabelecendo-se assim, comportamentos dinâmicos sociais e tecnológicos para que possam aproveitar, de uma forma eficiente, os recursos computacionais disponíveis [Ellis91;Greif88].

O objetivo desta dissertação consiste em descrever aspectos de desenvolvimento da Camada de Suporte para Aplicações Cooperativas, intitulada Camada Groupware, a qual fornece serviços e interfaces para o estabelecimento, manutenção e encerramento de aplicações cooperativas genéricas.

Genericamente falando, segundo Greif [Greif88], uma camada cooperativa pode ser descrita como:

> *"sistema baseado em computador que apoia grupos de pessoas envolvidas em uma tarefa ou meta comum e que provê uma interface para um ambiente compartilhado"*

A autonomia dos membros participantes da cooperação constitui-se em uma das principais preocupações a serem alcançadas pelas camadas de suporte às atividades cooperativas. Tal autonomia é conquistada graças aos quesitos, dentre outros, de sincronismo, segurança, qualidade operacional, transparência e o próprio gerenciamento de sistemas distribuídos [Bannon91].

Os quesitos acima citados são cobertos pelos chamados ambientes de Processamento Distribuído Aberto [RM-ODPa,b,c,d], abrangendo todas as principais características que proporcionam a dinamicidade de atuação dos membros envolvidos [Loyolla94a], sumarizando as idéias de transparência de distribuição, separação dos componentes entre os vários sistemas distribuídos envolvidos, heterogeneidade, flexibilidades para configuração, noções de federação [Herbert92], qualidade de serviço e a utilização de políticas contratuais [Coulson92].

Tratando-se de ações entre membros compartilhando um objetivo comum, estas ações poderão ocorrer, temporalmente falando, de forma assíncrona ou síncrona. Um compartilhamento síncrono se caracteriza quando as ações ocorrem simultaneamente, enquanto no assíncrono não há a necessidade de que a cooperação se manifeste em um mesmo momento. Portanto, um

ambiente cooperativo pode ser híbrido em relação ao aspeto temporal, ou seja, pode englobar mecanismos tanto para ações síncronas quanto para as assíncronas [Williams92].

A modelagem das aplicações cooperativas requer metodologias incrementais e iterativas devido à complexidade incorporada pelos ambientes distribuídos. As técnicas que envolvem a Orientação a Objetos [Rumbaugh91] possibilitam facilidades para o desenvolvimento de tais ambientes por integrarem as noções atribuídas ao contexto de objetos, tais como as de encapsulamento e reusabilidade.

Unindo-se às próprias metodologias de Orientação a Objetos, a identificação dos módulos que integrarão uma plataforma cooperativa é auxiliada pela análise do problema segundo os pontos-de-vista cobertos pelo Modelo de Referência ODP [Tschammer93; Linington92].

O desenvolvimento da Camada *Groupware* foi motivado pelo projeto da Plataforma Multiware [Loyolla94b]. A Plataforma Multiware visa auxiliar o desenvolvimento e fornecer suporte à nova geração de aplicações, a qual incorpora a necessidade de manipulação de informações de caráter multimídia sobre ambientes computacionais distribuídos e heterogêneos.

Para garantir os quesitos requeridos pelas aplicações cooperativas, a Plataforma Multiware é provida por três camadas básicas:

- Camada Básica de Software e Hardware: engloba as funcionalidades exportadas pelos sistemas operacionais e protocolos de comunicação
- Camada Middleware: fornece mecanismos de manipulação e armazenamento de informações multimídia em tempo real, assim como incorpora as funcionalidades ODP e de implementações comerciais de sistemas distribuídos [Loyolla94b; Mendes93; Tobar94]
- Camada Groupware: visa exportar serviços para o suporte a grupos [Loyolla94a]

Para descrever os aspectos envolvidos pela camada *Groupware*, esta dissertação apresenta-se dividida em nove capítulos, assim sumarizados.

O capítulo 2 realiza uma análise do conceito de Trabalho Cooperativo Suportado por Computador (CSCW - *Computer Supported Cooperative Work*), enfocando os aspectos humanos e tecnológicos envolvidos. Apresenta-se uma classificação temporal e geográficas das aplicações cooperativas, culminado com os cenários das principais aplicações existentes.

Em virtude da implementação alvo ser baseada em uma plataforma que integra as noções do Processamento Distribuído Aberto (ODP), o capítulo 3 apresenta os conceitos do Modelo de Referência ODP (RM-ODP). Introduz, ainda, a arquitetura CORBA, expondo, em seu final, a plataforma *Orbix r1.3.1*, que consiste em uma implementação comercial do modelo CORBA.

O capítulo 4 apresenta um estudo sobre a Técnica de Modelagem de Objetos (OMT), utilizada para a definição da arquitetura da camada Groupware.

A especificação da Camada Groupware é tratada no capítulo 5. Tal capítulo tem em seu início o contexto do projeto da Plataforma Multiware, enfocando a Camada de Serviços Exportáveis para o Suporte às Aplicações Cooperativas, denominada como Camada Groupware. Na descrição da Camada enfoca-se os seus serviços exportados para o suporte às atividades cooperativas.

O capítulo 6 consta do projeto da Camada Groupware, fazendo menção à arquitetura utilizada pela camada. Neste capítulo, são apresentadas as classes de objetos integrantes à Groupware, assim como uma breve descrição de sua funcionalidade.

Após a definição da arquitetura e dos objetos apresentados no capítulo 6, o capítulo 7 aborda os aspectos da implementação, enfocando os métodos pertencentes aos objetos através da descrição por meio de representação de eventos.

O capítulo 8 apresenta um exemplo de utilização da Camada Groupware. Para tanto, este capítulo aborda as etapas de iniciação da plataforma distribuída sobre a qual a Camada Groupware foi implementada e enfatiza os serviços fornecidos pela camada.

Por último, as conclusões e trabalhos futuros referentes ao projeto são descritos no capítulo 9.

Para auxiliar a manipulação da Camada Groupware, fazem parte da dissertação cinco anexos:

- Anexo A: Relação de arquivos pertencentes à Camada Groupware.
- Anexo B: Exemplo de um Makefile para integrar uma aplicação à Camada Groupware.
- Anexo C: Exemplo de uma aplicação cooperativa utilizando a Camada Groupware.
- Anexo D: Exemplo de registro no repositório de interfaces.

Capítulo 2 - Serviço Cooperativo Suportado por Computador

2.1 - Introdução

O Trabalho Cooperativo Suportado por Computador é um termo genérico que expressa o suporte ao trabalho de várias pessoas, organizadas em grupos, atuando em cooperação. Tal cooperação é conseguida graças às recentes tecnologias de *hardware* e *software*, em especial àquelas associadas com as redes de computadores e comunicação de informações multimídia.

Segundo Bannon [Bannon91], com o objetivo de fornecer facilidades para interfacer vários usuários em cooperação, o campo abrangido pelo *CSCW* deve englobar certos requesitos, tais como:

a) Suporte a articulações

Uma articulação consiste na coordenação de uma tarefa em particular, incluindo o gerenciamento das sub-tarefas, recuperação em face a possíveis erros e monitoramento dos recursos alocados [Gerson86]. As tarefas de articulação são necessárias para o estabelecimento e mediação entre indivíduos abrangidos pela ação cooperativa [Bannon91].

Qualquer proposta de suporte ao trabalho cooperativo envolve esforços no tocante ao controle e à mediação na interação de indivíduos. Tais esforços são aumentados quando o contexto da cooperação abrange uma distribuição, tanto geográfica quanto temporal, e requisitos de qualidade de serviço.

Há a necessidade de uma contínua negociação para o estabelecimento de regras contratuais entre as partes envolvidas na interação no início da transação. Tal necessidade origina-se devido à dificuldade para a definição correta do ambiente de cooperação, onde as articulações necessárias não podem ser previstas.

b) Compartilhamento de Informações

Um dos principais problemas dos sistemas de trabalho cooperativo é representado pela dificuldade de gerenciamento em relação ao compartilhamento de recursos e informações, uma vez que há um alto intercâmbio entre as pessoas envolvidas durante uma ação cooperativa. O referido problema agrava-se à medida que o grau de cooperativismo se acentua, pois aumenta-se, desta forma, a quantidade de articulações necessárias para o desenvolvimento da ação cooperativa.

Para tanto, uma camada de suporte para serviços cooperativos deverá fornecer mecanismos confiáveis e transparentes aos usuários. Tais mecanismos visam o controle de concorrência de acesso, recuperação e distribuição das informações remotamente localizadas.

c) Relação Tecnologia/Organização

Em grande parte, as alterações no comportamento de vida de uma população são causadas pelo desenvolvimento tecnológico e a sua conseqüente

introdução na sociedade. Devido à sua flexibilidade em relação às possibilidades de atuação junto aos diversos setores da sociedade, o computador se manifesta, na maioria dos casos, como o maior agente nas mudanças das relações sociais.

"Em todo momento, um sistema baseado em computador é introduzido em nosso trabalho, devendo este ser remodelado consciente ou inconscientemente. Nós não podemos escolher a ausência de impacto, apenas nós não podemos optar por sermos excessão de uma perspectiva. Nós podemos realizar escolhas conscientes cujos resultados e conseqüências nós previmos" [Bannon91].

Com este conceito, pode-se observar que, objetivando a cooperação entre pessoas, um sistema baseado em CSCW deve ser capaz de distinguir as formas de interação social, tendo como meta um baixo impacto sobre as relações de grupo.

2.2 - Caracterização dos Sistemas *CSCW*

Devido ao envolvimento de pessoas e recursos computacionais, sistemas baseados em *CSCW* podem ser discutidos sob dois aspectos distintos: aspectos humanos e aspectos tecnológicos [Wilson91].

A análise do trabalho cooperativo com o propósito de desenvolver modelos para as aplicações *CSCW* requer um estudo das características e formas de desenvolvimento de trabalhos cooperativos reais, uma vez que estas aplicações reúnem duas áreas do conhecimento: *social* e *tecnológica*.

Com relação ao aspecto social, a modelagem destes sistemas envolve características que são analizadas através da Sociologia, Psicologia e Ciência Organizacional.

Por outro lado, quanto ao aspecto tecnológico há que se considerar o auxílio apresentado pelo computador e pelos meios de comunicação.

Mais especificamente, a análise social deve se preocupar com os aspectos relacionados com o *Processo de Trabalho em Grupo*, enquando que a análise tecnológica deve se preocupar com o estudo da *tecnologia de software e hardware*, necessários para dar suporte ao trabalho cooperativo, considerando o computador como um meio de processamento de informação e as redes de computadores como um meio de troca destas.

Como o trabalho cooperativo em grupo está relacionado com as diversas formas de troca de idéias e informações, composição e reprodução do conhecimento pessoal e organizacional, formas de envolvimento de indivíduos do grupo em cada trabalho e organização do grupo, a tecnologia de software deve ser capaz de dar suporte para a mimetização da realidade deste tipo de trabalho através de métodos computacionais de informação, de forma que o espaço de trabalho e as comunicações sejam compartilhados, utilizando-se dos mecanismos de gerenciamento das atividades em grupo [Bannon91].

Outro aspecto a considerar na modelagem das aplicações *CSCW* é que se deve levar em consideração a dinâmica do processo de trabalho em grupo sob a óptica da tecnologia de software e hardware.

Cada aplicação *CSCW* deve poder dispor de todos estes aspectos tecnológicos, cada um apresentando um maior ou menor grau de importância em função de cada aplicação específica, mas todos colaborando para a composição do Processo de Trabalho em Grupo.

2.2.1 - Aspectos Humanos do Trabalho em Grupo

O conjunto de aspectos humanos para o trabalho em grupo, é resultado da combinação de três diferentes aspectos:

- individuais humanos
- organizacionais
- dinâmica de grupo

Pelo fato de não serem mutuamente excludentes, a referida combinação faz com que o processo de cooperação seja reconhecido.

2.2.1.1 - Aspecto Individual Humano

O aspecto individual humano visa a implantação de sistemas computacionais mais próximos à realidade defrontada pelos usuários em seus ambientes de trabalho. Aspectos individuais humanos estão intimamente relacionados com questões sociais, tais como a de distribuição das tarefas, formas de liderança e coordenação de reuniões.

Embora muito do trabalho cooperativo se desenvolva em grupo, o seu conteúdo normalmente baseia-se em trabalhos previamente desenvolvidos pelos agentes participantes do grupo, desta forma, as maneiras e métodos individuais sempre acabam por influir no desenvolvimento do trabalho em grupo.

As características individuais de comportamento levam à consideração de ao menos duas necessidades distintas a serem supridas pelos sistemas computacionais de trabalho em grupo:

- Interfaces para apresentação individual de informações, que permitam uma conversão simultânea de informação de acordo com certas preferências pessoais daqueles que compõem o grupo.
- Possibilidade de uso momentâneo e exclusivo do computador por parte de cada elemento do grupo, sem que isto seja compartilhado pelos demais elementos.

2.2.1.2 - Aspecto Organizacional

Os aspectos organizacionais estão relacionados com a estrutura e cultura da organização, com a representação do conhecimento organizacional e com as questões de gerenciamento [Wilson91].

Quanto à estrutura e cultura organizacional, os empreendimentos tendem a se comportar da mesma forma que quaisquer outras organizações humanas, freqüentemente atuando de forma inteligente. Tais empreendimentos podem, ainda, se adaptar à situações de mudanças e incertezas, aprendendo com suas próprias experiências e com a de outros, bem como incorporarem capacidade de planejar para um futuro razoavelmente incerto, de forma a explorar futuras oportunidades e evitar erros [Blanning92].

O conhecimento organizacional pode ser expresso através de conhecimento codificado e na forma heurística.

Por conhecimento codificado entende-se o conjunto das variadas formas de bases de dados contendo informações específicas, localizada e temporalmente identificadas, documentos contendo expressão contextualizada de um conjunto de dados, geralmente enfocando aspectos específicos de interesse de um grupo.

Por conhecimento heurístico organizacional entende-se como a filosofia e metodologia de trabalho do empreendimento.

Tanto o conhecimento humano como o organizacional podem ser afetados em sua origem pela própria estrutura organizacional da empresa, já que o estabelecimento de divisões e hierarquias funcionais definem a área de atuação de cada elemento do grupo. A forma de atuar de cada indivíduo participante de uma aplicação *CSCW* pode ser especificada a partir de experiências vividas por elementos dentro de organizações humanas.

Outro fator norteador da atuação de cada indivíduo participante da ação cooperativa, para o auxílio no modelamento de trabalho em grupo, é o conjunto de normas específicas a um dado trabalho. Estas normas, geralmente, são estabelecidas por um responsável pelo grupo (moderador ou coordenador), que também se responsabiliza pelo gerenciamento do mesmo. Este gerenciamento se refere tanto a pessoas quanto a recursos, ou ainda ao próprio trabalho de grupo como um todo.

Uma vez que o conjunto de informações em um empreendimento encontra-se distribuído ao longo de toda a sua extensão, cada membro do grupo deve ser capaz de poder obtê-las, parcial ou totalmente, quando necessárias. Para o acesso às informações devem ser considerados a sua localização, a forma em que se encontra, seu conteúdo, sua variação temporal e as restrições de acesso.

Os aspectos organizacionais citados levam à consideração de algumas necessidades a serem supridas pelos sistemas computacionais para dar suporte ao trabalho em grupo. Sob o ponto de vista computacional, uma plataforma cooperativa deve integrar diversos mecanismos, dentro os quais destacam-se:

- Disponibilidade de um repositório do conjunto do conhecimento organizacional e humano, de fácil acesso à informação desejada, com disponibilidade de formatos de conversão, tendo em vista a diversidade da forma na qual estas se encontram armazenadas.
- Disponibilidade, ao gerenciador do trabalho em grupo, de várias possibilidades de controle e andamento do trabalho em grupo, de forma que a escolha possa ser dinâmica, de acordo com as necessidades imediatas do empreendimento.

2.2.1.3 - Ponto de Vista da Dinâmica de Grupo

O ponto de vista da dinâmica de grupo está relacionada com o processo de colaboração dos indivíduos em um determinado grupo, ou seja, trata do desenvolvimento do trabalho, das formas de tomada de decisão e consulta aos participantes bem como do comportamento do grupo de forma global. O processo de colaboração depende de dois fatores principais: da composição do grupo e do próprio trabalho atuando sob o escopo do cooperativismo a ser desenvolvido.

A composição do grupo está relacionada com questões do tipo: seu tamanho, agentes que o compõe (personalização dos componentes), sua existência temporal (duração prevista) e distribuição espacial (proximidade dos agentes).

O tamanho do grupo e a personalização dos seus participantes é que determinam os recursos de comunicação necessários à aplicação bem como os mecanismos de colaboração a serem empregados.

O tipo de trabalho a ser realizado pelo grupo se constitui em fator preponderante na determinação das ferramentas computacionais que deverão estar disponíveis aos participantes do mesmo.

Como fatores que afetam diretamento a dinâmica do processo de colaboração destacam-se: a previsão da existência do grupo e da sua duração bem como a forma pela qual se processa sua extinção; as questões relativas à distribuição geográfica e temporal dos indivíduos cooperantes em um grupo.

2.2.2 - Aspectos Tecnológicos

Os aspectos relacionados com a tecnologia de hardware e de software necessários para dar suporte ao trabalho cooperativo podem ser avaliados considerando-se quatro sistemas distintos [Wilson91;Rodden92]: apoio às atividades de grupo, apoio ao espaço de trabalho compartilhado, apoio ao compartilhamento de informações e apoio à comunicação. Estes sistemas se complementam na constituição de um sistema complexo de apoio à cooperação.

Os aspectos tecnógicos visam oferecer mecanismos necessários para que os aspectos humanos sejam cobertos pelas plataformas cooperativas.

2.2.2.1 - Suporte às Atividades de Grupo

O suporte às atividades de grupo refere-se às formas de organizar e controlar o processo de composição do trabalho em grupo, independentemente do trabalho a ser desenvolvido.

Neste contexto, é necessário o estabelecimento prévio das funções e papéis dos indivíduos participantes do grupo. Os papéis podem ser alterados no decorrer do processo de colaboração.

Uma vez reconhecidas as funções, com base em metas que se pretende atingir, é possível estabelecer procedimentos que devam ser seguidos pelos agentes que participam do trabalho cooperativo.

2.2.2.2 - Suporte ao Espaço de Trabalho Compartilhado

O suporte ao espaço de trabalho compartilhado consiste em permitir que diferentes agentes participantes de uma aplicação vejam e trabalhem sincronizadamente em um mesmo espaço eletrônico de trabalho.

Sistemas desta natureza correspondem a Sistemas de Compartilhamento Remoto de Telas, Sistemas de Interação Face-a-Face e Quadro Branco Inteligente [Wilson91].

A característica de sistemas de compatilhamento remoto de telas é que estes permitem que parte da tela da estação de trabalho de um participante seja replicada nas estações dos demais participantes da aplicação e se constitui em uma área comum de trabalho, de forma que qualquer alteração introduzida na mesma é visualizada por todos.

Quanto aos sistemas de interação face-a-face, a sua característica principal é de possibitar a composição de uma única tela pública apresentada nas telas das

estações de trabalhos dos indivíduos que participam da mesma aplicação. Esta tela tem a função de exibir informações úteis para a ação cooperativa.

Por último, quadro branco inteligente se caracteriza pelo suporte às atividades assíncronas, fornecendo mecanismos de inclusão de informação, desenho, reorganização de informação e impressão de documentos eletronicamente armazenados.

2.2.2.3 - Suporte ao Compartilhamento de Informação

O suporte ao compartilhamento de informações refere-se às formas de permitir que usuários vejam e trabalhem, geralmente ao mesmo tempo, em um mesmo conjunto de informações [Wilson91].

Os principais sistemas de suporte ao compartilhamento de informações caracterizam-se por sua forte orientação a objetos, habilitando o acesso de vários grupos de trabalho a grandes bases de informação multi-usuário que suportem acesso e manipulação de documentos hipertexto ou multimídia.

Entre as principais facilidades oferecidas por estes sistemas pode-se citar a possibilidade do estabelecimento de nichos de trabalho, interligados através de gerenciadores de documentos, que provêm a atualização e acompanhamento das versões do documento compartilhado.

2.2.2.4 - Suporte à Comunicação

O suporte à comunicação refere-se às diferentes formas de receber e enviar mensagens para locais diferentes. Engloba os aspectos tecnológicos de digitalização, formatação, compactação, transmissão e recuperação da informação, tanto a nível de hardware como de software [Navarro92].

2.2.2.5 - Suporte a Transparência

Atividades cooperativas estão, na maioria dos casos, associadas com o conceito de distribuição do grupo, onde os seus membros podem participar a partir de diversos lugares e diferentes horários, interfaces distintas, etc [Navarro92].

Em termos de cooperativismo, define-se dimensão como o conjunto de fatores que envolvem os aspectos organizacionais, temporais, físicos, linguísticos e culturais. Serviços CSCW têm a obrigação de ocultar as dimensões que dificultam as ações de interação entre diversos membros de um grupo, introduzindo o conceito de transparência no relacionamento entre eles.

Devido a tais fatores, surgiram diversas necessidades de transparências, como: transparência de organização, transparência de tempo, transparência de visão e transparência de atividade.

2.2.3 - Classificação Espaço/Tempo

O trabalho cooperativo apresenta características importantes de tempo e espaço, que determinam, em uma primeira instância, a demanda de recursos necessários para dar suporte a sua existência.

Em termos temporais, a interação entre os indivíduos participantes da cooperação pode ser de forma isócrona ou anisócrona. Os membros dentro de uma

interação isócrona se interagem no mesmo espaço de tempo, enquanto que na anisócrona, a interação pode ocorrer em tempos distintos.

Por outro lado, em termos espaciais os indivíduos podem estar localizados em um mesmo espaço geográfico ou em localizações distintas. Estas possibilidades estão sintetizadas através da matriz espaço-tempo [Ellis91] que é apresentada na figura 2.1.

Tempo / espaço	**mesmo tempo**	**tempo diferente**
mesmo lugar	*interação face-a-face*	*interação anisócrona*
lugar diferente	*interação isócrona distribuída*	*interação anisócrona distribuída*

Figura 2.1 - Relação espaço/tempo

Com relação aos recursos necessários, localização geográfica e ação temporal, as aplicações *CSCW* podem ser agrupadas em quatro classes de sistemas computacionais, que são:

- *Sistemas de Mensagens*
- *Conferência Computadorizada*
- *Salas de Encontro*
- *Sistemas de Co-Autoria e Argumentação*

A figura 2.2 mostra estas formas de cooperação para sistemas baseados em CSCW em relação à classificação temporal. O eixo vertical representa a distinção do caráter isócrono em relação ao anisócrono. Na mesma figura, nota-se que os sistemas de co-editoração e argumentação podem ser tanto anisócronos quanto isócrono, os demais serviços são estritamente pertencentes a uma única característica temporal.

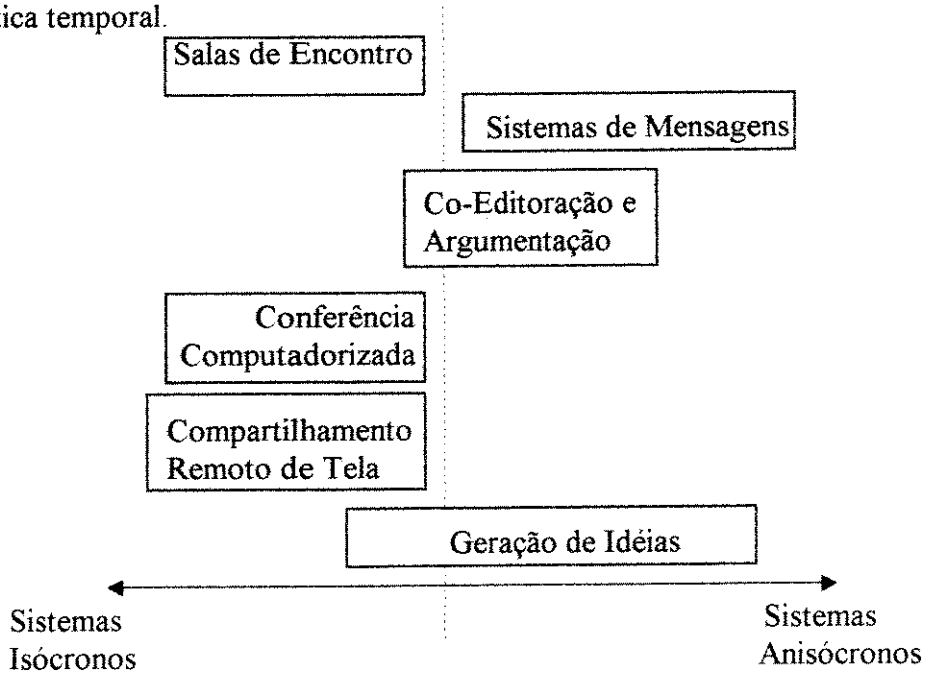


Figura 2.2 - Divisão das classes de serviços CSCW por localização temporal

As mesmas classes de serviços CSCW são ilustrados na figura 2.3, porém distinguidas pela distribuição geográfica. Os eixos verticais definem as fronteiras entre cada aspecto de localização geográfica, podendo alguns serviços possuirem mais de uma característica quanto ao aspecto de localização.

Na figura 2.3, são definidos os seguintes termos:

- Locais: Todos os usuários localizados em um mesmo espaço geográfico. Exemplo: sala de encontro.
- Virtualmente locais: Há um compartilhamento de recursos dito como locais, porém os membros não necessitam estar em um mesmo espaço físico. Exemplo: vídeo-conferência.
- Localmente Remotos: Sistemas que necessitam ter transferência de informações em alta velocidade, como em sistemas em tempo real. Exemplo: co-editoração.
- Remotos: Existe uma mínima cooperação direta entre os membros. Exemplo: sistemas de mensagem.

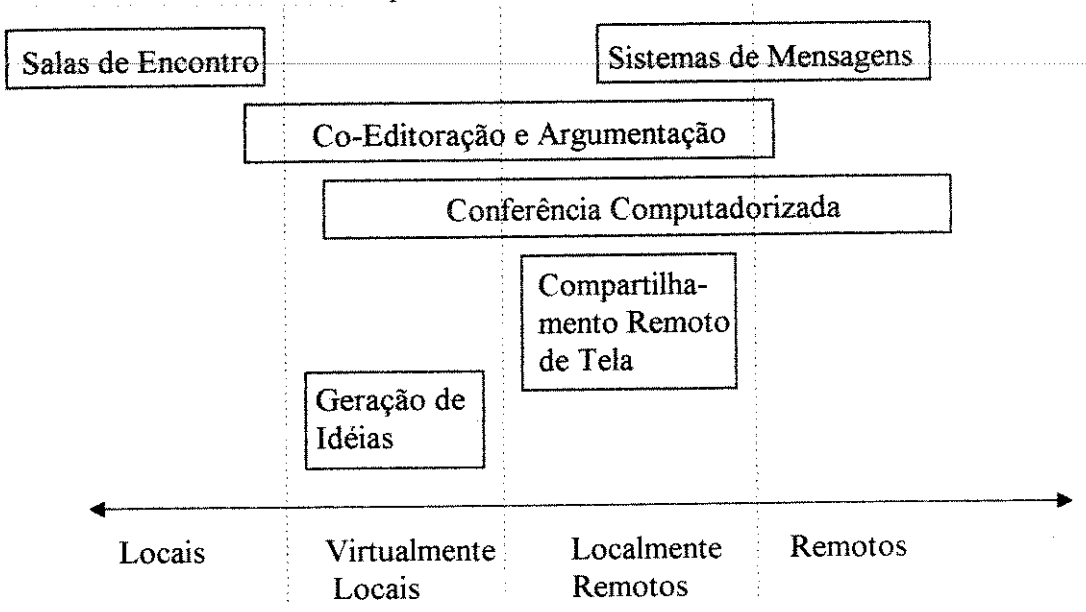


Figura 2.3 - Divisão das classes de serviços CSCW por localização geográfica

As características geográficas nem sempre representam com fidelidade as diferentes formas de aplicação sobre trabalho cooperativo existentes. A dificuldade para a perfeita caracterização é conseqüência de que as aplicações cooperativas, freqüentemente, incorporam características dinâmicas, compreendendo mais de uma classe geográfica, constituindo-se em sistemas mistos. Nestes sistemas é possível se distiguir apenas a cooperação isócrona ou anisócrona entre os usuários que participam do grupo.

Sistemas de Mensagens

Os sistemas de mensagens são descendentes dos primeiros programas de correio eletrônico, atualmente normalizados através dos padrões CCITT - X400 (Message Handling System - MHS). Caracterizam-se por operações anisócronas, onde uma dada estrutura é anexada à própria mensagem a ser transmitida, sendo

que cada sistema de mensagem pode fazer uso de um formato particular para transmitir a informação.

Tais sistemas devem fornecer mecanismos para a composição, envio e exibição de mensagens multimídia, a partir de um usuário para um ou vários destinatários.

Conferência Computadorizada

A vídeo conferência se constitui em uma aplicação isócrona que permite dois ou mais *usuários*, isolados fisicamente entre si, se interconectarem por intermédio de uma rede de alto desempenho (por exemplo *B-ISDN*).

Em resumo, a conferência computadorizada baseia-se na realização de uma conferência baseada apenas nas estações de trabalho como ponto de contato entre os participantes.

Sala de Encontro

A sala de Encontro é considerada como um sistema formado por um grupo de usuários participantes, localizados ao mesmo tempo em uma mesma sala, compartilhando recursos comuns, geralmente representados por uma tela onde são projetadas as informações do usuário correntemente na ação de explanação de algum ponto da reunião.

Cada usuário dispõe de um terminal de um computador central ou de uma máquina conectada a uma rede de computadores.

O gerenciamento da aplicação é realizado através de um terminal de controle que tem a função de dirigir a seqüência de acesso e votação de cada usuário do sistema. Tal gerenciamento pode ocorrer de forma automática, especificada no estabelecimento do grupo, ou pela ação direta do mediador do encontro no decorrer da ação cooperativa.

Neste tipo de aplicação, tanto as informações como o seu processamento podem ser distribuídos. Caracteriza-se por um sistema que requer elevadas taxas de transmissão, uma vez que a quantidade de informações de dados, vídeo e áudio que trafegam pode ser muito alta.

Tendo em vista a possibilidade de descentralização de informações, também há que se levar em consideração as questões de sincronismo na manipulação das informações.

Sistemas de Co-Autoria e Argumentação

Os Sistemas de Co-Autoria e Argumentação são sistemas de suporte à argumentação e votação por parte de vários usuários, localizados distribuídos ou em um mesmo espaço geográfico. Cada usuário atua argumentando e votando sobre um certo assunto de interesse, manipulando informações multimídia ou "convencionais".

Estas aplicações se caracterizam pelo isocronismo dos eventos gerados pelos agentes participantes, sua distribuição espacial e heterogênea.

Tendo em vista o uso compartilhado das informações, o aspecto da proteção de acesso às mesmas neste tipo de aplicação é fundamental. Mecanismos

de recuperação de informação devem ser implementados, como forma de prevenção, nos casos em que não se pode manter a qualidade do serviço.

Outro aspecto a considerar é a possiblidade da agregação dinâmica e autônoma de outros indivíduos em um trabalho cooperativo já em andamento.

Geração de Idéias

Esta aplicação, cuja operação é semelhante a uma "caixa de sugestões", permite que diversos *usuários* tenham acesso somente para escrita dentro de uma área destinada ao armazenamento de dados, armazenando as suas críticas, idéias ou segestões a um certo assunto previamente estabelecido. Os dados armazenados podem ser utilizados posteriormente em outras aplicações.

Compartilhamento Remoto de Tela

O compartilhamento remoto de tela consiste na possibilidade de que partes públicas da tela da estação de trabalho de um certo usuário sejam visualizadas simultaneamente por outros usuários igualmente pertencentes ao grupo.

2.3 - Cenários Exemplos de Aplicações CSCW

Com o objetivo de se definir os objetos básicos a serem propostos e que deverão compor a camada *GROUPWARE*, para darem suporte as aplicações *CSCW* [Rosenberg92;Lee93], apresenta-se a seguir uma análise da dinâmica envolvida em cada uma das aplicações *CSCW*, considerando ainda que estas se desenvolverão em um Ambiente de Processamento Aberto e Distribuído.

Entende-se como *usuários*, indivíduos que possuem uma área de trabalho, definida em um sistema computacional de uma instituição, e capazes de iniciarem uma determinada aplicação *CSCW* ou participarem de alguma que venha a se estabelecer ou que já esteja em andamento.

Define-se *recursos computacionais* os componentes computacionais, disponíveis via rede de comunicação de dados ou particularmente a um usuário, destinados ao processamento das informação.

Sistemas de Mensagens

O processo de geração de mensagens pode ser caracterizado pelas seguintes operações: definição do(s) destinatário(s), composição da mensagem a ser transmitida e a sua transmissão propriamente dita.

No destinatário, a mensagem recebida é armazenada em arquivo com formatação definida por algum protocolo para tal finalidade e pode ser utilizada como elemento de entrada para a aplicação CSCW. Entende-se aqui, que o conteúdo da mensagem recebida pode se constituir em apontadores para as partes que integram um objeto composto do tipo multimídia, cuja recuperação somente se processa durante a atividade de visualização.

Conferência Computadorizada

Na fase de iniciação os *usuários* são ligados entre si e os *recursos* são alocados de acordo com a qualidade de serviço especificada pelo requisitante da aplicação ou através de acordo entre os usuários participantes.

No início da conferência computadorizada, também chamada de vídeo-conferência, deve ser estabelecida uma política administrativa, de modo a possibilitar a inclusão de novos participantes ou a exclusão dos já integrados, bem como permitir mudanças das características individuais ou globais no decorrer da aplicação.

Entende-se como características individuais, aquelas que estão relacionadas com as necessidades de cada usuário em um determinado instante, enquanto que as globais se referem àquelas de interesse do grupo cooperativo e que estão diretamente relacionadas com a qualidade do serviço durante a evolução da aplicação.

Durante a evolução da aplicação, métodos automáticos devem ser executados a fim de: a) assegurar o padrão da qualidade global, uma vez que qualquer alteração local poderá proporcionar alterações de interesse coletivo; b) verificar a existência de pedidos de entrada de novos participantes, bem como o desligamento dos já integrados; c) averiguar a ocorrência de falhas relacionadas aos recursos e às vias de comunicação, procedendo o possível restabelecimento do ambiente.

A finalização da aplicação somente se processa mediante solicitação do *usuário* requisitante ou devido à queda da Qualidade de Serviço associada a níveis inaceitáveis, onde são desfeitas todas conexões, destruindo-se todos os canais *usuários-usuários* e *usuários-recursos* existentes.

Sala de Encontro

Neste tipo de aplicação, um objeto gerenciador deve existir, tendo como função a conexão de todos os participantes aos *recursos* necessários, de modo que estes possam ser manipulados de forma concorrente.

No que se refere à política administrativa da cooperação entre os *usuários* participantes, esse serviço deve contar com as mesmas características da vídeo-conferência, isto é, seleção dos *recursos*/qualidades e entrada/saída de *usuários*.

Para este tipo de aplicação, as conexões ocorrem somente entre *usuários* e *recursos*, que são estabelecidas no momento da requisição do serviço, não havendo, portanto, a interconexão formal entre os próprios *usuários*.

Durante o andamento da aplicação, devem ser executados métodos automáticos para tratamento da ocorrência de falhas e o possível restabelecimento das conexões quando necessário.

A finalização da aplicação se processa mediante solicitação do *usuário* requisitante, que tem o papel de moderador, ou ocasionada devido à queda da QoS a níveis não aceitáveis, ocasião em que ocorrem as desconexões dos canais previamente estabelecidos.

Co-Autoria

Estabelecida a seleção das pessoas que poderão participar da co-autoria e a definição do assunto a ser tratado, é criada uma área área de trabalho, que conterá todas as informações relativas ao documento em questão, garantindo-se o controle de versões e a revisão dos pontos de conexão dos documentos parciais.

O acesso por parte de cada usuário pode ocorrer de forma isolada, a partir de uma requisição individual de participação, desde que o requisitante esteja na lista dos possíveis participantes da co-editoração.

O processamento de alteração do conteúdo dos documentos co-editorados deverá ser realizado mediante um controle de concorrência por intermédio de um objeto gerenciador, o qual se constitui no sistema de monitoramento dos participantes conectados. O controle de versões, revisão dos pontos de conexão entre os documentos ou partes dos mesmos bem como a associação entre estes, também têm que ser garantidos.

Assim como as outras aplicações *CSCW*, a co-autoria deve contar com mecanismos que proporcionem uma melhor segurança em relação ao acesso às informações, bem como ao controle de falhas, permitindo que seja garantido o restabelecimento caso haja ocorrência de alguma.

De modo semelhante ao acesso, o fechamento do elo de ligação entre os usuários e os recursos deve ser processado de forma individual.

Geração de Idéias

A geração de idéias se processa de forma semelhante à co-editoração, diferenciando apenas no modo de manipulação da base de dados.

Os serviços operam de forma pública, sem estabelecer uma lista de possíveis participantes. Somente as operações de escrita (a fim de proporcionar inclusões de novas idéias) e de leitura (para verificar as idéias já armazenadas) podem ser acessadas, de forma excludente.

Assim como a co-editoração, a geração de idéias funciona de modo anisócrono e requer o uso de mecanismos de controle de concorrência para o acesso à base de dados.

As fases de conexão, manutenção e término são praticamente idênticas à co-editoração, apenas diferenciando na ausência do objeto gerenciador, uma vez que não há a necessidade de se proceder o monitoramento entre participantes.

Compartilhamento Remoto de Tela

Após estabelecida uma política de acesso, os *usuários* são agrupados em níveis hierárquicos e funcionais. Aqueles que têm o direito de visualizar partes públicas de telas alheias poderão fazê-lo em sua estação de trabalho através da conexão com os *usuários* selecionados.

Para a visualização, um *usuário* que possui o privilégio de observar as partes públicas telas de outros *usuários*, requisita a aplicação de Compartilhamento Remoto de Tela. Neste momento estabelecem-se conexões entre o solicitante e aqueles que terão suas telas visualizadas. As conexões são representadas pela

abertura de canais de comunicação por onde as informações compartilhadas trafegam.

As conexões são desfeitas apenas quando um pedido de desconexão é solicitado pelo requisitante do serviço.

Capítulo 3 - Processamento Distribuído Aberto (ODP)

3.1 - Introdução

Com o intuito de integrar diversos sistemas computacionais, um grande esforço é dispendido no tocante à padronização dos ambientes distribuídos e heterogêneos, objetivando um intercâmbio e um processamento das informações mais flexíveis, consistentes, confiáveis e eficientes.

Esse capítulo enfoca a padronização para Processamento Distribuído Aberto desenvolvida em conjunto pelas organizações *ISO* (ISO/IEC 10746) e *ITU-T* (X.901 a X.904) e, em seguida, é apresentado uma plataforma *ODP* orientada para objetos, confeccionada pelo *OMG* (Object Management Group).

3.2 - Padronização *RM-ODP*

O modelo de referência ODP (*RM-ODP; Reference Model ODP*) busca estabelecer padrões para permitir a interconexão entre ambientes heterogêneos e pertencentes a vários domínios organizacionais, atendendo as características e dificuldades gerais dos sistemas distribuídos: concorrência, dificuldade de reconhecimento do estado global do sistema, falhas locais, relação assíncrona dos processos e a possibilidade de processamento remoto [RM-ODPa,b,c,d].

Além das características acima, a padronização ODP cobre as seguintes propriedades:

- mobilidade: possibilidade de migração de dados, processos e até de usuários por entre os nós de um sistema distribuído
- autonomia: as partes integrantes de um sistema são autônomas, porém interrelacionadas
- criação de federações: união de sistemas de domínios distintos para estabelecer uma meta em comum
- manipulação da qualidade de serviço: a introdução dessa característica permite configurar a qualidade de um sistema afim de caracterizar seu comportamento frente a, por exemplo, falhas dos recursos locais associados, tempo e custos de utilização de recursos, dentre outras
- gerenciamento: capacidade de controlar e monitorar recursos associados com o sistema, de modo a permitir a cobertura da qualidade de serviços, políticas de manutenção e demais componentes de configuração
- transparência: mascarar os aspectos e falhas decorrentes de um sistema distribuído em relação às aplicação e usuários

Com a finalidade de cobrir os aspectos de distribuição como proposto pelo modelo *ODP*, um sistema distribuído pode ser especificado segundo cinco pontos

de vista, distintos e não independentes entre si, os quais expressam características particulares de tal sistema como um todo.

3.2.1 - Pontos de Vista do Modelo de Referência *ODP*

Um ponto de vista auxilia na obtenção das necessidades específicas de um sistema, reconhecendo propriedades relevantes e descartando aquelas irrelevantes sob um certo aspecto de apreciação [Griethuysen92], provendo mecanismos de abstração para o problema.

Os pontos de vista podem ser analogamente comparados com as etapas cobertas pela engenharia de software, conforme é ilustrado a figura 3.1 [Raymond93].

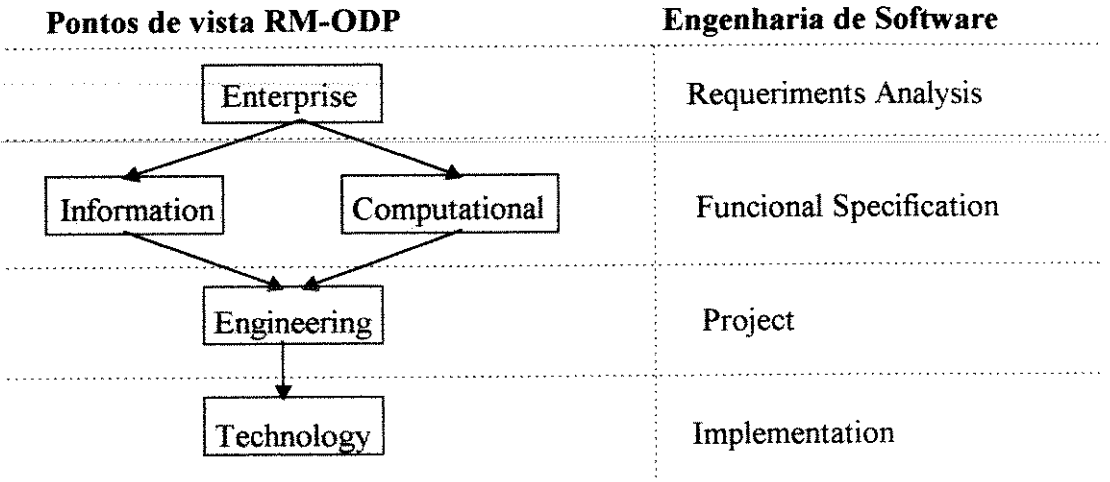


Figura 3.1 - Comparação Pontos de Vista (RM-ODP) e Engenharia de Software

A seguir são descritos os pontos de vista ODP segundo o Modelo de Referência ODP [RM-ODPa].

3.2.1.1 - Ponto de Vista Empresarial

O ponto de vista empresarial relaciona-se diretamente com o levantamento das necessidades dos usuários dentro de um certo sistema, estabelecendo propósitos, escopo e políticas de utilização.

Para a descrição dos pontos de vista empresarial, será utilizado o exemplo ilustrado na figura 3.2. Nesta figura é abordada a filosofia de cliente-servidor, onde um *componente servidor* é manipulado por um elemento humano *operador* e o componente *cliente* é manipulado por dois *operadores*. Os componentes *cliente* e *servidor* são interconectados por intermédio dos serviços de uma rede de comunicação de dados.

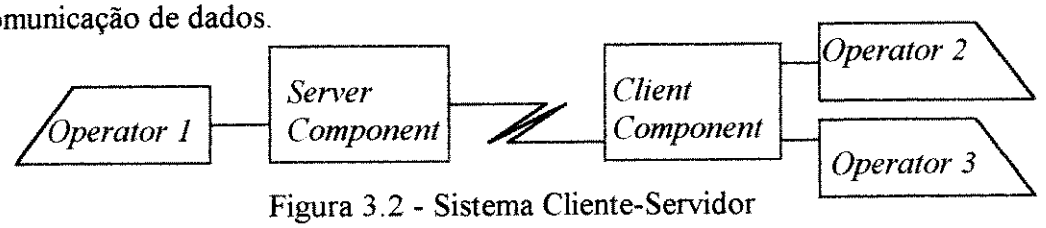


Figura 3.2 - Sistema Cliente-Servidor

Por intermédio do ponto de vista empresarial, o sistema da figura 3.2 é visto em termos de:

- o objetivo do sistema
- as informações necessárias por cada operador do sistema
- a interação do operador 1 com o componente servidor
- a interação dos operadores 2 e 3 com o componente cliente
- a interação entre os componentes servidor e cliente
- as políticas, incluindo as de segurança, que gerenciam as interações entre os componentes
- as qualidades de serviços requisitadas pelos operadores 1, 2 e 3.

As políticas sociais e organizacionais dentro da empresa, atuantes sobre as estruturas e requisitos levantados segundo esse ponto de vista, podem ser classificados como:

- agentes: estruturas ativas, as quais poderão dar início às ações executáveis e modificar o estado de estruturas já criadas
- artefatos: estruturas passivas, onde somente serão alvo das ações dos agentes, ou seja, não tem poder de manipulação sobre as demais estruturas
- comunidades: conjunto formado por agentes e artefatos

Sobre os agentes, artefatos e comunidades poderão atuar políticas em termos de permissão, proibição e obrigações. Exemplificando com uma rede de computadores, temos:

Como exemplo, pode-se classificar os componentes de uma rede de computadores a nível de agentes, artefatos e comunidades e estabelecer políticas sobre os mesmos, assim como especificado abaixo:

- agentes: administrador da rede, usuários
- artefato: arquivos no servidor de arquivos
- comunidade: a própria rede como um todo
- políticas de permissão: um usuário somente pode modificar os seus próprios arquivos
- políticas de proibição: um usuário não tem acesso a arquivos de outros usuários
- obrigações: o administrador possui como obrigações cadastrar usuários e instalar softwares

3.2.1.2 - Ponto de Vista de Informação

Sob este ponto de vista, as aplicações atuantes sobre uma certa informação são tratadas como funções do tipo *black-box*. Tem o objetivo de reconhecer somente a estrutura da informação, conjuntamente com as regras de validade, e o relacionamento entre aplicações.

O levantamento das informações é realizado segundo os seguintes tópicos:

- estrutura das informações necessitadas pela empresa
- regras que definem o relacionamento entre elementos da informação
- alterações sobre a informação após a atuação de um certo processamento
- atributos de qualidade
- fluxo das informações
- visibilidade das informações e aplicações em relação aos usuários da empresa

3.2.1.3 - Ponto de Vista Computacional

O ponto de vista computacional é responsável pelo projeto da arquitetura da aplicação ODP, não havendo preocupação, nesta fase, do equipamento e dos recursos disponíveis na empresa.

Para estabelecer tal arquitetura, o ponto de vista computacional baseia-se nas definições de objetos como elementos básicos de processamento, que são estruturas capazes de encapsular atributos (dados) e métodos (alteradores de estado) responsáveis pela sua caracterização. O relacionamento entre objetos é feito por intermédio de interfaces, podendo ser múltiplas para cada objeto.

Sendo assim, o presente ponto de vista tem como objetivo definir, a nível funcional, os objetos inseridos em uma aplicação ODP assim como os seus interrelacionamentos dentro da mesma.

A definição mais detalhada de objeto está presente no item 3.2.2.1 deste capítulo.

3.2.1.4 - Ponto de Vista de Engenharia

Após definidos os objetos integrantes de uma certa aplicação, surge a necessidade de associá-los às características técnicas pertinentes com a própria implementação, como processadores, memória, rede de comunicação, mecanismos de controle e de transparência, etc. O ponto de vista de engenharia cobre os seguintes aspectos:

- especificação concreta dos tipos abstratos de dados identificados no ponto de vista computacional
- identificação dos requisitos de qualidade de serviço
- especificação do comportamento dos objetos envolvidos na aplicação ODP

3.2.1.5 - Ponto de Vista Tecnológico

Descreve a aplicação sob o ponto de vista de implementação, levando-se em conta a disponibilidade de hardware e produtos de software.

Em resumo, o ponto de vista tecnológico visa:

- escolher o hardware e software de acordo com as características levantadas pelos outros pontos de vista do Modelo de Referência ODP
- identificar o impacto ocasionado pela escolha tecnológica
- levantar informações sobre o comportamento do sistema sob aspectos tecnológicos envolvidos com o ambiente computacional

3.2.2 - Aspectos Gerais de Modelagem

Toda modelagem no universo ODP é puramente baseada nos conceitos de objetos e, segundo o modelo descritivo do RM-ODP, tais conceitos podem ser divididos em três conjuntos:

- conceitos de modelagem básica
- conceitos de especificação
- conceitos arquiteturais

3.2.2.1 - Conceitos de Modelagem Básica

Neste nível de conceituação, são introduzidos a definização de objeto e os termos a ele associados, tais como encapsulamento e abstração,comportamento e estado, *interface* e modelagem de objetos em ODP.

3.2.2.1.1 - Conceito de Objetos

Define-se objeto como sendo uma estrutura dotada de uma identificação, a qual é única em seu domínio, que encapsula atributos e métodos que poderão ser exportados mediante solicitações externas de outros objetos [Takahashi90].

A relação de exportação/importação ocorre a nível de interfaces dos objetos envolvidos, a qual contém a prototipação e referências às suas estruturas internas (figura 3.3).

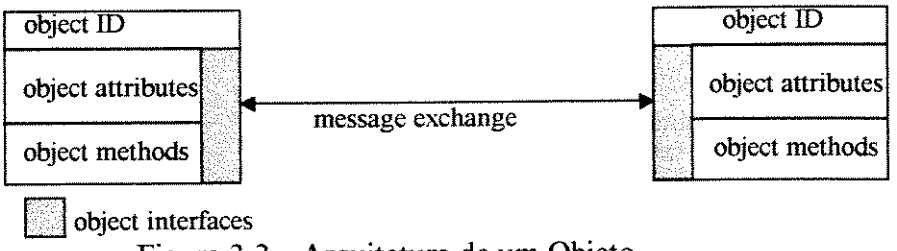


Figura 3.3 - Arquitetura de um Objeto

3.2.2.1.2 - Encapsulamento e Abstração

Por intermédio do encapsulamento, é conseguido um mecanismo de controle de acesso aos dados e métodos contidos em um objeto, onde tal acesso é realizado atrav´s de interfaces. Já a abstração assegura ocultamento dos detalhes

encapsulados em relação aos demais objetos pertencentes à aplicação ou que compartilham o mesmo domínio.

Desta forma, consegue uma maior autonomia dos objetos e uma melhor modularização do sistema, podendo, portanto, englobar várias tecnologias e plataformas para a suas implementações, favorecendo a idéia de heterogeneidade de ambientes.

3.2.2.1.3 - Comportamento e Estado

Define-se como comportamento de um objeto, o conjunto de todas as ações passíveis de acontecimento dentro de um objeto, as quais poderão modificar o seu estado que representa sua situação em um dado instante. Sendo o comportamento um conjunto de ações, este pode ser definido como um mapeamento das mudanças de estado dentro do objeto.

3.2.2.1.4 - Interface

Interface representa o conjunto de ações encapsuladas em um objeto e observáveis externamente pelos demais objetos. Assim sendo, constitui-se em uma abstração do comportamento de um objeto obtida pela observação das suas ações que poderão ser invocadas por outros objetos pertencentes a um certo domínio.

É permissível a um objeto ODP possuir várias interfaces, dividindo o seu conjunto de interação em vários sub-conjuntos seguindo separação e distribuição funcionais.

A identificação e distinção de interfaces existentes são realizadas por intermédio de identificadores de interfaces [Rudkin92].

3.2.2.1.5 - Modelagem de Objetos em ODP

O modelo de objetos é essencial para descrever, especificar e projetar sistemas ODP, onde a abstração propicia a heterogeneidade dos mesmos, além de permitir a substituição ou modificação de objetos sem a necessidade de modificações no ambiente.

Objetos podem possuir uma granularidade, comportamentos, nível interno de paralelismo e número de interações entre si altamente arbitrários

3.2.2.2 - Conceitos de Especificação

Os conceitos de especificação estão vinculados à escolha correta de uma linguagem de especificação para a modelagem do sistema ODP. O fato da possibilidade de aplicação dos conceitos sobre as linguagens de especificação ou de programação não representa a dependência destes em relação às próprias linguagens.

Os conceitos são representados pelas idéias de composição e decomposição, *templates*, papéis (*roles*), tipos e classes, hierarquia classe/sub-classe e hierarquia classe base/classe derivada.

3.2.2.2.1 - Composição e Decomposição

Os processos de composição e decomposição são úteis durante a etapa de refinamento do projeto, onde um objeto pode ser decomposto em vários objetos menos complexos segundo os diferentes níveis de abstração alcançado.

A partir da decomposição de um objeto, consegue-se a identificação de novos objetos mais simples e específicos em relação ao objeto originador. Neste caso, classes de objetos de níveis mais altos são configuradas a partir de objetos das camadas inferiores de forma ortogonal (figura 3.4).

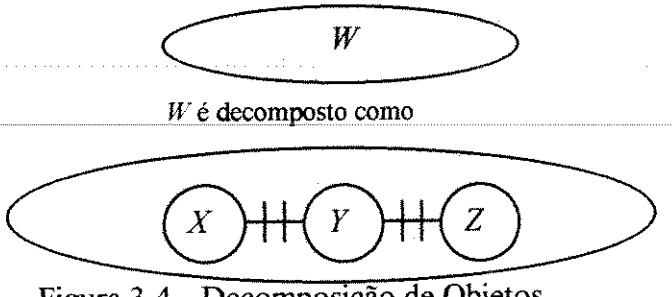


Figura 3.4 - Decomposição de Objetos

Na figura acima, o objeto *W* pode ser decomposto, por refinamentos sucessivos, nos objetos *X*, *Y* e *Z*.

Por outro lado, a composição de objetos pode resultar em uma visão ampla e única do sistema, onde uma árvore hierárquica é tratada como um único objeto de alto nível.

Na figura 3.5 é mostrado uma composição de dois objetos, denominados de *X* e *Y*. Onde *X* deve interagir perfeitamente com *Y*, fazendo com que as interações entre os dois objetos tornem-se escondidas e manipuladas como sendo ações internas de um objeto composto.

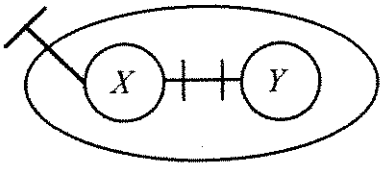


Figura 3.5 - Composição de Objetos

3.2.2.2.2 - Templates

Os *templates* são estruturas que caracterizam objetos de mesmo comportamento, visando descrever os traços comuns de objetos similares e delimitar as suas diferenças por intermédio da descrição dos parâmetros de estado e operações.

Os *templates* possuem as informações suficientes e necessárias, em um determinado nível de abstração, para possibilitar instanciações de novos objetos, estabelecendo, dessa maneira, o estado inicial dos mesmos.

A figura 3.6 reporta o caso em que um conjunto de objetos, em sua configuração inicial, é obtido a partir da instanciação de um único *template*.

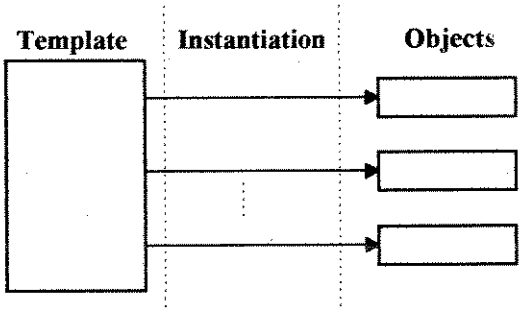


Figura 3.6 - Instanciação de Objetos a partir de *templates*

3.2.2.2.3 - Papéis

Representa a função desempenhada por um objeto dentro de uma interação, revelando o motivo pelo qual o mesmo é participante. Um objeto pode assumir diversos papéis, distintos ou não, em diversas interações em um dado momento. Os papéis mais típicos são representados pelo binômio produtor/consumidor.

A figura 3.7 mostra o objeto *A* assumindo dois papéis distintos: o papel de produtor em relação à interação com o objeto *B* e o de consumidor em relação ao objeto *C*.

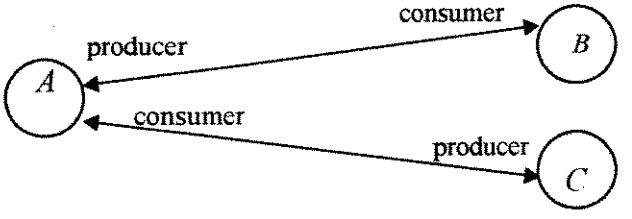


Figura 3.7 - Papéis desempenhados pelos objetos nas interações

3.2.2.2.4 - Tipos e Classes

Denomina-se tipo como sendo uma propriedade de uma coleção de objetos. Um objeto é de um certo tipo se ele satisfizer a propriedade relacionada com o referido tipo, os demais atributos não prescritos podem ser diferentes.

Objetos de mesmo tipo podem ser agrupados de modo a construirem uma classe de objetos. Portanto, classe é uma coleção de objetos pertencentes a um mesmo tipo.

3.2.2.2.5 - Hierarquia Classe/Sub-Classe

A noção de classe e tipo já embutem a conceituação de sub-tipo e sub-classe. A distinção entre classe e sub-classe pode ser feita analogamente àquela entre conjunto e sub-conjunto, onde uma classe é uma sub-classe de outra se, e

somente se, a primeira é um sub-conjunto da segunda e, um tipo é sub-tipo de outro se os predicados do primeiro implicam os predicados do segundo.

3.2.2.2.6 - Hierarquia Classe Base/Classe Derivada

A hierarquia entre classe base e classe derivada é baseada na noção de modificação de *templates* ou de classes. Refinamento de um *template* tem como conseqüência alterações em relação aos valores iniciais de um conjunto de objetos. Modificações a nível de *templates* são denominadas de *herança incremental* [RM-ODPb].

Assim como pode-se modificar a estrutura dos *templates*, tal conceito pode ser aplicado a nível da implementação das classes, tendo como um dos maiores méritos a possibilidade de reutilização do código. Nesse caso, ocorre uma herança de implementação.

3.2.2.3 - Conceitos Arquiteturais

Conceitos arquiteturais relacionam-se com a descrição e organização dos sistemas distribuídos, buscando características relevantes às distintas concepções dos sistemas atuais. Os conceitos arquiteturais englobam as definições de nome, grupos, domínios, contrato, *liasion* e *binding*, este último sendo distinguido entre *binding explícito* e *binding implícito* [RM-ODPb].

3.2.2.3.1 - Nome

A utilização do nome é associada com a identificação e acesso de compontes pertencentes a um sistema distribuído.

Tratando-se de componentes dentro de um ambiente distribuído, podendo este estar relacionado com os conceitos heterogeneidade, autonomia e federação, necessita-se de uma política de nomeação de tais componentes totalmente dependente do contexto no qual estão associados. Estas políticas são necessárias para o estabelecimento de uma estratégia de resolução de nomes.

3.2.2.3.2 - Grupos

O Modelo de Referência ODP define genericamente o conceito de grupo como sendo um conjunto de objetos com características estruturais ou funcionais comuns.

3.2.2.3.3 - Domínios

Assim como o conceito de grupos, o RM-ODP introduz o conceito de domínios de um modo altamente genérico, permitindo assim que sejam cobertas as noções de autonomia, autoridade e controle presentes nos sistemas distribuídos.

O RM-ODP define que objetos podem pertencer a certo domínio desde que seus comportamentos sejam controlados por uma autoridade em comum.

3.2.2.3.4 - Contrato

O conceito de contrato foi gerado com o intuito de estabelecer normas de cooperação entre dois objetos, caracterizando-as e regulamentando-as. Tem como fundamento a introdução das idéias de obrigações e expectativas em torno de uma cooperação, incorporando uma ação dinâmica na mesma. Os contratos podem, por exemplo, reger normas que especificam os papéis dos objetos, a qualidade de serviço esperada e os comportamentos que os invalidam.

3.2.2.3.5 - *Liaison* e *Binding*

A cooperação entre objetos é realizada segundo dois conceitos contratuais: o *liaison* e o *binding*. Para o estabelecimento da conexão entre dois objetos, ocorre a etapa de *binding*, que corresponde ao acordo contratual para que na próxima fase, representada pelo *liaison*, possa realmente ocorrer a cooperação regida pelo contrato previamente estabelecido.

Pode-se concluir que um *binding* produz um *liaison* e que um objeto pode estar envolvido em vários *liasions* simultaneamente.

A etapa de *binding* pode se dar por duas diferentes formas: o *binding* explícito e o *binding* implícito, conforme descrito abaixo.

3.2.2.3.5.1 - *Binding* Explícito

O *binding* explícito se caracteriza pela instanciação de um objeto que tem a função de encapsular os mecanismos de controle da cooperação, fornecendo um caráter manipulável ao *binding*. Tal objeto recebe como argumentos o próprio tipo e o conjunto inicial de identificadores de interface. Caso a etapa ocorra com sucesso, o objeto envolvido recebe um ponteiro para a interface de controle do *binding*, a qual contém, no mínimo, uma operação: o *unbind*, responsável pelo término da transação.

A figura 3.8 ilustra o relacionamento de dois objetos por intermédio de um *binding* explícito.

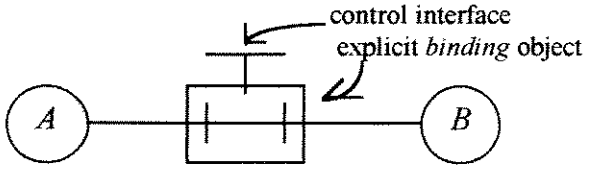


Figura 3.8 - Conexão de dois objeto (*A* e *B*) com *binding* explícito

3.2.2.3.5.2 - *Binding* Implícito

Diferente do explícito, o *binding* implícito não tem a ação de um objeto de controle, onde a sua manutenção é mantida até o final da transmissão da informação ou o seu tempo é estipulado pelo contrato inicial (figura 3.9).

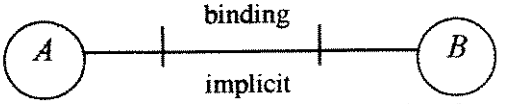


Figura 3.9 - Conexão de dois objeto (*A* e *B*) com *binding* implícito

Para melhor ilustrar o processo do *binding* implícito, serão utilizados os diagramas exibidos nas figuras de 3.10 a 3.17, que mostram as interações entre interfaces.

1- Considere um objeto cliente *A* em seu estado inicial (figura 3.10).

Figura 3.10 - Objeto cliente em seu estado inicial

2 - Instanciação de um objeto *B* que, no caso, desempenhará o papel de servidor (figura 3.11).

Figura 3.11 - Instanciação de um objeto servidor

3 - Associado ao objeto *B*, foi criada uma interface *x* do tipo *I1* (denotada por *x:I1*). Pela notação, a seta voltada para o objeto associado indica a função de servidor (figura 3.12).

Figura 3.12 - Criação da interface *x:I1*

4 - O objeto *B* anuncia, por exemplo para um *trader*, a existência da interface *x* do tipo *I1* (figura 3.13). *Trader* representa um objeto que contém um mapeamento de serviços exportados por objetos servidores para que sejam contactados pelos clientes. Em suma, os objetos servidores fazem anúncio de seus serviços por intermédio de um *trader* e, os objetos clientes podem escolher os serviços exportados pelos servidores contactando a lista presente no *trader*. O *trader* ODP também é alvo de padronização, porém tal padronização não é parte integrante do RM-ODP [Bearman93].

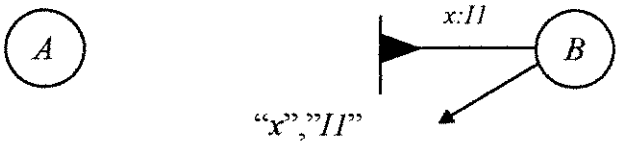


Figura 3.13 - Anúncio da existência de *x:I1*

5 - Criação, por parte do objeto *A*, da interface *y* do tipo *I2* (figura 3.14).

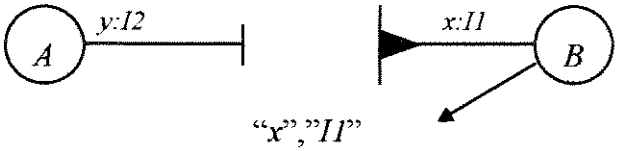


Figura 3.14 - Criação da interface *y:I2* por parte de *A*

6 - O objeto *A* anuncia que deseja a comunicação com algum objeto do tipo *I* (figura 3.15).

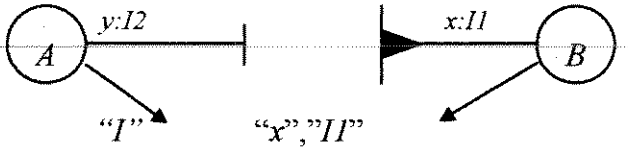


Figura 3.15 - Objeto *A* anuncia necessidade de comunicação com alguma interface

7 - O objeto *A* descobre, por intermédio de um *trader* por exemplo, a existência de uma interface *x* do tipo *I1*, representando um sub-tipo de *I* (figura 3.16).

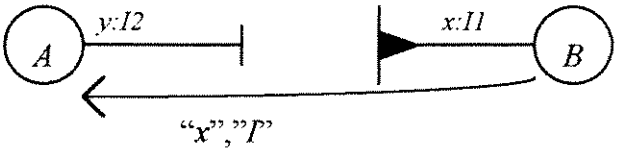


Figura 3.16 - Encontro da interface *x:I* por *A*

8 - Após o *binding*, os objetos *A* e *B* estão realizando a cooperação por intemédio de suas interfaces, representando a etapa de *liaison* (figura 3.17).

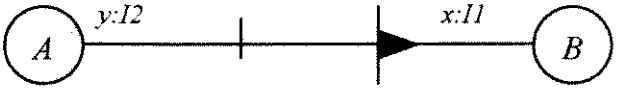


Figura 3.17 - *Liaison* envolvendo as interfaces *x:I1* e *y:I2*

3.3 - Arquitetura *CORBA*

Com o intuito de desenvolver um padrão de Processamento Distribuído Aberto, porém totalmente modelado segundo o paradigma de objetos, consolidou-se um consórcio entre empresas de renome, referenciado como *Object Management Group (OMG)*, que definiu a arquitetura *CORBA* (Common Object Request Broker Architecture).

A filosofia básica da *CORBA* consiste no intercâmbio de mensagens, de forma transparente, entre objetos de um mesmo domínio ou de domínios distintos.

Um domínio representa, neste caso, um conjunto computacional qualquer, regido sob uma certa arquitetura e um certo sistema operacional [OMG93].

3.3.1 - Estrutura do Object Requester Broker

Um *Object Request Broker* (ORB) é uma plataforma computacional responsável por suprir todos os mecanismos necessários para:

- achar a implementação de um objeto servidor a partir da requisição de um cliente
- preparar o servidor para receber requisições dos objetos clientes
- gerenciar toda comunicação necessária para o intercâmbio entre o cliente e o servidor

Devido ao ORB, a interface que o cliente visualiza é totalmente independente da localização geográfica, da linguagem utilizada para a implementação do servidor ou de qualquer outro aspecto não referente ao escopo da interface.

Para descrever a ação do ORB será utilizada a idéia da relação entre cliente e servidor, onde as fases que constituem o estabelecimento da conexão entre estes são de responsabilidade do ORB, que possui todos os mecanismos necessários para tal.

Denota-se cliente como sendo a entidade que deseja importar uma operação pertencente a um outro objeto qualquer, e como servidor aquele possuidor da implementação (código e dados) que será exportada.

Integram a fase de estabelecimento da conexão entre um cliente e um servidor desempenhadas pelo ORB as seguintes etapas:

- requisição da execução do método, por parte do cliente, ao ORB
- localização, dentro do conjunto de servidores, da implementação requisitada
- preparação do servidor para receber o pedido do cliente
- comunicação do pedido do cliente ao servidor

Abaixo, estão listadas algumas definições presentes no modelo CORBA [OMG96]:

- Objetos são estruturas identificáveis e encapsuladas, podendo fornecer um ou mais serviços. A sua identificação é feita por uma referência de objeto, única para um certo domínio.
- Requisição representa uma invocação de serviço por parte de um cliente. O contexto da requisição está presente na mensagem trocada pelas interfaces dos objetos. Caso um erro ocorra durante a transação, o objeto servidor retorna uma mensagem de *exceção*.
- Os objetos são classificados segundo o seu tipo, o qual atua sobre as características de compartilhamento.

- Interfaces especificam o conjunto de possíveis operações que um cliente pode solicitar de um objeto servidor. Desta forma, a interface simboliza o comportamento do objeto.
- Os serviços passíveis de serem evocados constituem as operações, representadas pelos seus identificadores (nomes). Os argumentos de entrada, saída e exceções de uma operação são especificadas por intermédio de sua assinatura.
- IDL (Interface Definition Language) consiste em uma linguagem para a definição de objetos. Tal mapeamento é independente de linguagens de programação e das características das plataformas computacionais (sistema operacional e hardware).

A figura 3.18 ilustra o relacionamento cliente/ORB/servidor. A interface estabelecida entre o cliente e o ORB ou entre o ORB e o servidor é totalmente independente da linguagem de programação, localização, estruturação do objeto ou qualquer outro tópico não relacionado diretamente ao conceito de interface.

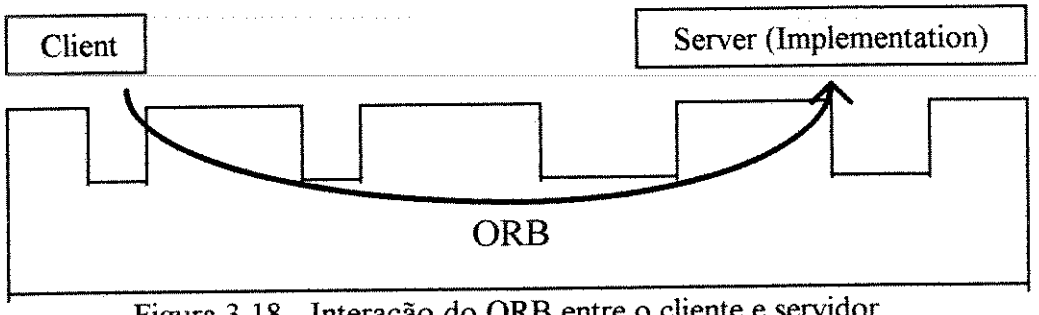
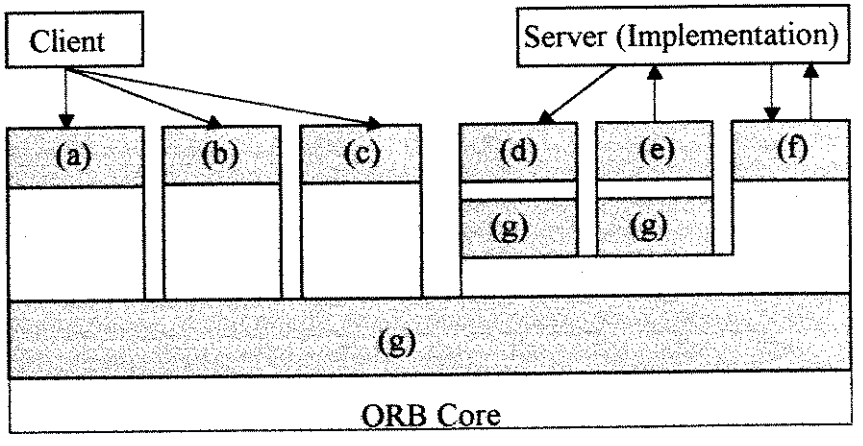


Figura 3.18 - Interação do ORB entre o cliente e servidor

Para a cooperação entre objetos, o modelo CORBA disponibiliza facilidades para a definição de objetos, objetivando um simples mapeamento entre os vários modelos presentes no suporte das implementações.

A arquitetura ORB é representada na figura 3.19.



(a) Static Invocation Interface
(b) *Stubs*
(c) ORB Interface
(d) Static *Skeleton*
(e) Dynamic *Skeleton*
(f) Object Adapter
(g) ORB Core Interface

Figura 3.19 - Arquitetura ORB

Em relação à figura 3.19 as interfaces descritas por intermédio das letras (a) até (g) possuem as seguintes características:

interfaces	descrição	tipo de interface
(a)(c)(e)	Interf. idênticas para todas as implementações ORB	interface *up-call*
(f) (b)(d) (g)	Possíveis adaptadores de objetos múltiplos *Stubs* e *Skeletons* para cada tipo de objeto Interfaces dependentes do ORB	interface de chamada normal

Um objeto cliente pode requisitar serviços de objetos servidores por intermédio de interfaces de invocação dinâmica ou estática, tendo ainda a possibilidade de interagir com o núcleo ORB para solicitar alguns mecanismos neste presentes.

A invocação estática se baseia na utilização de interfaces *stub*, definidas por intermédio de uma linguagem de definição de interfaces (IDL). As interfaces *stub* são específicas a cada objeto provedor de serviços.

Por sua vez, a invocação dinâmica ocorre quando a definição de acesso se manifesta em tempo de execução, mediante a importação de interfaces contidas em um repositório de interfaces.

Após a solicitação do cliente, o ORB tem a função de localizar e contactar a interface que contém o serviço solicitado. Seguida a contactação, ocorre a transferência do controle para a implementação do serviço por intermédio da interface para os métodos, denominada como *skeleton IDL*. Por sua vez, assim como o cliente, a implementação do servidor pode também acessar o núcleo ORB, porém via um Adaptador de Objetos.

A etapa de confecção das interfaces, implementações e repositórios é mostrada na figura 3.20, onde nota-se que as interfaces são originárias de compilação de arquivos IDL e as implementações a partir de linguagens de programação.

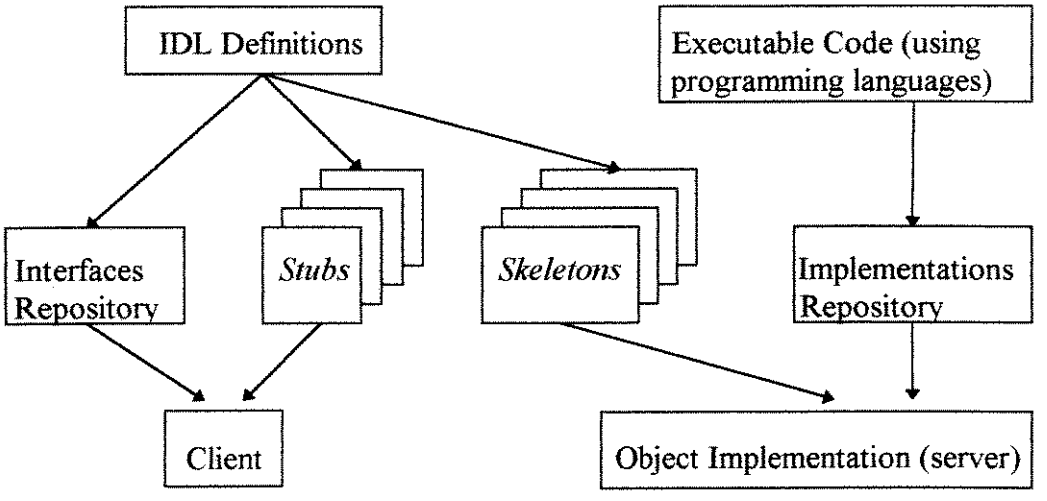


Figura 3.20 - Relação entre definições, implementações e repositórios

Nos ítens a seguir são descritas as definições dos componentes integrantes da arquitetura CORBA.

3.3.1.1 - Núcleo ORB

Responsável pela interação entre objetos clientes e servidores. Fornece serviços de negociação, acesso, conexão, manutenção e desconexão da cooperação entre ambos.

A estruturação baseada na interação puramente de interfaces faz com que sejam mascaradas todas as divergências em relação ao equipamento e à linguagem de programação utilizados.

3.3.1.2 - Implementações de Objetos

Corresponde ao conjunto de código (métodos) e dados (atributos) pertinente a um objeto com um comportamento específico. Geralmente implementações de objeto são não dependentes do ORB, porém podem selecionar interfaces dependentes do ORB através dos Adaptadores de Objetos.

3.3.1.3 - Referências de Objetos

Referência do objeto serve para identificá-lo dentro de um domínio ORB. Caso o sistema seja constituído por diversos ORB, estes deverão possuir a mesma política de mapeamento de linguagem para que possa haver a perfeita interação entre eles, independentemente das linguagens de programação utilizadas.

3.3.1.4 - Linguagem de Definição de Interfaces (IDL)

Durante a fase de definição dos objetos utiliza-se, para a especificação das interfaces, uma linguagem *IDL* (Interface Definition Language), contendo uma gramática que abrange os mecanismos necessários para as operações em um ambiente distribuído aberto.

Por intermédio das definições escritas em IDL, consegue-se abstrair o mapeamento dos objetos implementados em diversos tipos de linguagens de programação, orientados ou não a objetos.

3.3.1.5 - Interface de Invocação Dinâmica

Utilizada pelos objetos ORB clientes para a especificação e possível acesso dinâmico das operações dos objetos servidores. A construção do pedido é feita em tempo de execução de acordo com as necessidades correntes dos clientes. Para tanto, o código do cliente fornece ao núcleo ORB informações sobre a operação a ser requisitada. A estrutura das informações (assinatura) pode ser previamente obtida através, por exemplo, de um repositório de interface.

3.3.1.6 - *Stubs*

Rotinas associadas aos mecanismos de evocação estática, onde um cliente incorpora ao seu código as operações particulares de objetos ORB servidores, para que seja realizado o acesso durante a execução.

3.3.1.7 - *Skeleton* Estático

Corresponde às interfaces, geralmente associadas ao tipo *up-call*, para os métodos implementados pelos objetos servidores ORB.

Um *skeleton* estático tem a função interfacear objetos clientes às implementações de objetos servidores para que possam ser invocados estaticamente os métodos exportados pelos servidores.

Desta forma, um *skeleton* estático está para o servidor assim como o *stub* está para o objeto cliente.

3.3.1.8 - *Skeleton* Dinâmico

Assim como o *skeleton* estático, o dinâmico também visa o interfaceamento entre os objetos clientes e servidores, porém a invocação dos métodos baseia-se em mecanismos dinâmicos. A comunicação entre os objeto cliente e a implementação de objetos é feita entre a interface de invocação dinâmica e o *skeleton* dinâmico, respectivamente.

3.3.1.9 - Adaptadores de Objetos

Realiza o interfaceamento entre as implementações dos objetos e o núcleo ORB para o acesso de serviços destinados à manutenção do ambiente de distribuição. Em geral, os serviços fornecidos pelo ORB incluem:

- geração e interpretação de referências de objeto
- invocação de métodos
- segurança de interações
- ativação e desativação de objetos e implementações
- mapeamento das referências de objetos
- registro de implementações

3.3.1.10 - Interface ORB

A interface ORB tem a função de fornecer um elo de ligação entre os objetos ORB e o núcleo ORB, suportando operações básicas não cobertas pelos adaptadores de objeto, *stubs*, *skeletons* ou pela interface de invocação dinâmica.

3.3.1.11 - Repositório de Interface

Relacionado com operações contratuais dinâmicas. Armazena as informações geradas a partir de um compilador IDL, constituídas nas assinaturas dos serviços exportáveis pelos objetos servidores.

É, a partir do repositório de interface, que um cliente consegue distinguir e requisitar as operações mantidas pelos objetos ORB. No mecanismo de invocação dinâmica, tais informações são desconhecidas, por parte do cliente, em tempo de compilação.

3.3.1.12 - Repositório de Implementação

Responsável por fornecer informações necessárias ao ORB para efetuar a localização e ativação das implementações dos objetos quando requeridas pelos clientes.

Dentre suas informações destacam-se aquelas específicas ao ambiente operacional, tais como equipamento e sistema operacional, informações sobre a instalação das implementações dos objetos e políticas relacionadas com a ativação, manutenção e liberação das mesmas, segurança, recursos alocados, etc.

3.4 - Plataforma *Orbix*

A plataforma *Orbix* consiste em uma implementação comercial do modelo *CORBA*, onde os objetos *Orbix* se interagem por intermédio de suas interfaces, possuindo uma única referência dentro do domínio coberto pela plataforma [Orbix95a,b].

Devido à possibilidade de implementação dos objetos em diversas linguagens, surge a necessidade de compatibilizá-los mediante a utilização de interfaces. Para tanto, utiliza-se uma linguagem de definição de objetos (IDL) que, no caso do Orbix, segue o padrão sugerido pela OMG.

Após criadas as interfaces necessárias para a obtenção dos objetos clientes e servidores, a próxima etapa a ser seguida consiste na compilação do arquivo IDL para produzir classes *C++* denominadas, segundo a terminologia *Orbix*, como *C++ IDL*. Tal processo é mostrado na figura 3.21.

Figura 3.21 - Etapa de Criação *IDL C++*

3.4.1 - Conceitos *Orbix*

3.4.1.1 - Proxy

Devido às características de distribuição, uma referência a um objeto pode se dar local ou remotamente. Neste último caso, o ponteiro fará menção a um objeto *proxy*, que consiste na representação de um objeto remoto e oferece todos os mecanismos de comunicação para tal tipo de localização.

A figura 3.22 ilustra uma cooperação remota de objetos, realizada, portanto, mediante a utilização de um *proxy*. Nota-se nesta figura que toda a comunicação entre o cliente e o servidor é "*forwarded*" pelo objeto *proxy*.

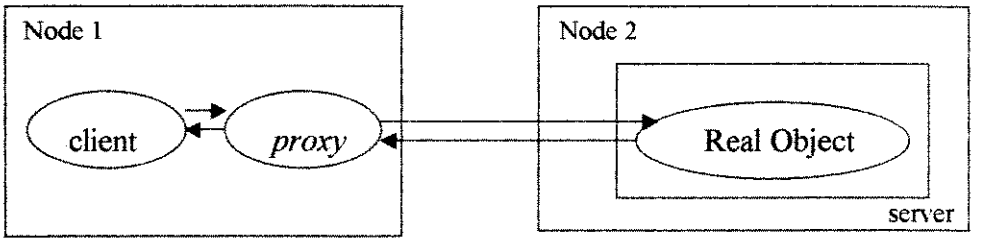


Figura 3.22 - Relacionamento remoto de dois objetos por meio de um *proxy*

3.4.1.2 - Modos de Ativação

Após implementado o servidor, o programador necessita registrá-lo em um Repositório de Implementação para que seja encontrado por algum cliente.

O Orbix permite diferentes mecanismos de ativação do servidor, especificados no momento de seu registro no repositório:

- modo de ativação compartilhado: todos os processos com o mesmo nome de servidor em uma certa máquina, são gerenciados por um processo comum.
- modo de ativação não compartilhado: cada processo de um servidor é registrado no repositório, havendo um gerenciamento individual sobre os mesmos.
- modo de ativação por método: assim como o compartilhado, cada processo é registrado no repositório, porém são permitidas chamadas inter-processos. Neste tipo de ativação, ocorre a destruição de processo assim que a operação foi completada.

Cada um dos modos de ativação acima descritos, permite ainda dois tipos de ativação:

- por cliente: a ativação de um mesmo servidor por diferentes clientes, tem como conseqüência a ativação de vários processos (um para cada usuário).
- por múltiplos clientes: a ativação de um mesmo servidor por diferentes clientes, resultará no compartilhamento dos processos.

3.4.1.3 - Threads

A versão utilizada (release 1.3.1) admite somente um *thread* por processo, resultando em somente um pedido ativo por processo [Tanenbaum92].

3.4.1.4 - Implementação de Interfaces

A implementação dos objetos (código *C++*) deve conter o código previsto na definição de suas interfaces (*C++ IDL*). O Orbix suporta dois mecanismos de implementação: *BOAImpl Approach* e o *TIE Approach*.

3.4.1.4.1 - *BOAImpl* Approach

A implementação de uma interface utilizando o *BOA* (Basic Object Adapter) resultará em uma classe *C++ IDL* que na verdade representa uma sub-classe *BOAImpl*. Tal sub-classe irá incorporar as características especificadas no mapeamento *IDL* do usuário e também as contidas na classe *BOA*.

Fazem parte da classe *BOA*, os métodos necessários para:

- Gerar e interpretar referências de objetos.
- Autenticação das chamadas às referências de objetos.
- Ativação e desativação de implementações
- Ativação e desativação de objetos
- Métodos de invocação através de *skeletons*

3.4.1.4.2 - *TIE* Approach

O *TIE Approach* representa um mecanismo de implementação de interfaces na qual as classes de implementação criadas a partir das classes *IDL* não apresentam herança de outras classes.

Quando utilizado o *TIE Approach*, ocorre uma completa separação entre a classe de interface e a classe de implementação gerada a partir de uma *IDL* e uma linguagem de programação, respectivamente.

Por intermédio do *TIE* o programador assegura a não utilização do *BOA*, permitindo uma maior flexibilidade e provendo facilidades não disponíveis no *BOA*, tais como a definição de "filtros a nível de objeto".

Capítulo 4 - Técnica de Modelagem de Objetos (OMT)

4.1 - Introdução

Entende-se orientação por objeto como o termo que denota a organização do problema como sendo uma coleção de objetos, os quais são estruturas que incorporam estruturas dados e comportamentos.

A principal característica da manipulação de objetos é a abstração de diferentes conceitos usados por pessoas distintas para representar o seu conhecimento, incorporando um detalhamento hierárquico e real do mundo a ser questionado.

Diversos pesquisadores têm se concentrado para a formalização de metodologias orientadas para objeto, merecendo destaque os trabalhos de Rumbaugh [Rumbaugh91], Booch [Booch91], Shlaer-Mellor [Shlaer88] e Coad-Yourdon [Coad91]. Atualmente, metodologias orientadas em objetos encontram um vasto campo de aplicação em diversas indústrias, dentre as quais incluem-se as de telecomunicações [Yamazaki93].

O objetivo desse capítulo é descrever uma metodologia de projeto orientada para objetos desenvolvida por James Rumbaugh [Rumbaugh91], utilizada para auxiliar o projeto da Camada Groupware. Tal metodologia denominada de *Object Modeling Technique* (OMT) consiste em uma representação gráfica, independente das linguagens de programação, capaz de auxiliar na análise dos requisitos do problema e no projeto de uma solução ao referido problema. O OMT integra três modelos: modelo de objeto, que é o responsável pela descrição dos objetos inseridos no sistema e seus relacionamentos; modelo dinâmico, referenciando as interações entre os objetos; e o modelo funcional, que descreve o fluxo dos dados e suas alterações.

4.2 - Conceitos Básicos

Metodologias orientadas para objetos têm a necessidade de incorporar mecanismos para que os conceitos diretamente relacionados com o paradigma de objetos possam ser menipulados. Dentre tais conceitos, destacam-se: identidade, classificação, polimorfismo, herança, abstração e encapsulamento.

Neste tópico serão apresentados os conceitos básicos acima citados, que serão utilizados no decorrer da explanação sobre a metodologia OMT.

4.2.1 - Identidade

A identidade expressa uma referência para um objeto, única dentro de um domínio. O objeto é defido como sendo uma estrutura independente, composta por um estado interno e um comportamento.

O estado interno é representado pelo conjunto de atributos (variáveis e constantes) internos que caracterizam o objeto em função de seus valores assumidos em um dado momento. O comportamento relaciona-se com os métodos (procedimentos/funções) que são capazes de alterar o estado do próprio objeto, enviar e receber mensagens trocadas entre os outros objetos.

4.2.2 - Classe

Classes têm o poder de agrupar objetos de estruturas e comportamentos idênticos. Portanto, define-se classe como sendo uma abstração de um conjunto de objetos, descrevendo informações importantes e necessárias para o domínio do problema. Desta forma, pode-se descrever objetos a partir da definição de classes: um objeto é uma instanciação de uma classe. Objetos resultantes de instanciação de uma mesma classe podem compartilhar atributos e operações entre si.

4.2.3 - Polimorfismo

O polimorfismo é a concepção de que diversos objetos podem conter atributos e métodos iguais quanto ao seu nome, porém diferentes em sua ação, ou seja, manisfestam diferentemente em cada objeto, de acordo com o papel estabelecido para este.

4.2.4 - Herança

A noção de herança é originária de refinamentos das classes, onde a organização hierárquica vertical permite que aquelas colocadas nas camadas inferiores (classes derivadas) herdem atributos e métodos daquelas localizadas nas superiores (classes bases). A identificação da hierarquia é realizada mediante refinamentos sucessivos, de forma a especializar as estruturas gerais conhecidas, conforme ilustra a figura 4.1.

A especialização é caracterizado pela introdução de características adicionais aos objetos descendentes de maneira que o comportamento destes seja mais específico para uma certa função. Em contrapartida, uma classe base é mais genérica, pois abriga comportamento utilizado pelas várias classes derivadas especializadas.

Ainda em relação à figura 4.1, é apresentado um exemplo de superclasse e subclasse, onde a superclasse é aquela mais geral (classe 1) e as subclasses são aquelas mais especializadas (classes 1.2, 1.1.1, 1.1.3 e 1.1.2.1). Devido à hierarquização, as mais classes especializadas podem utilizar os atributos e métodos das classes mais gerais localizadas na verticalização da organização.

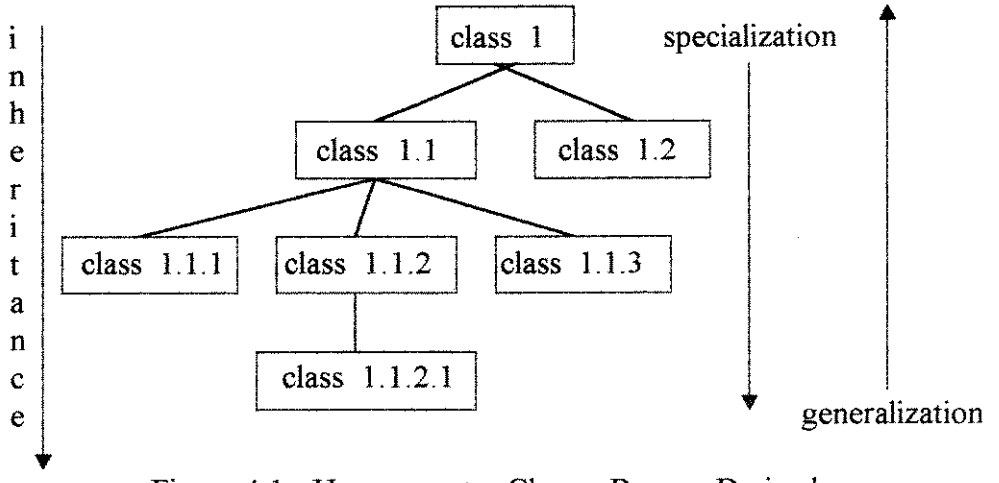


Figura 4.1 - Herança entre Classes Bases e Derivadas

A nível de implementação, o conceito de herança é extremamente útil na reutilização de código devido ao agrupamento de características comuns aos objetos, permitindo uma fácil alteração e adaptação entre as classes.

4.2.5 - Abstração

O termo abstrair significa identificar as características essenciais de uma certa entidade (domínio do problema) e deprezar aquelas não importantes para tal caracterização. A abstração a nível de orientação para objeto é essencial para a sua modelagem, afim de incorporar somente um conjunto finito e pequeno (se possível) de particularidades do mundo real enfocado.

4.2.6 - Encapsulamento

O processo de encapsulamento consiste em separar as informações externas das internas, ocultando aquelas que não podem ser acessadas por outros objetos. Desta forma, impede-se que o efeito das alterações internas se propague aos demais módulos do sistema.

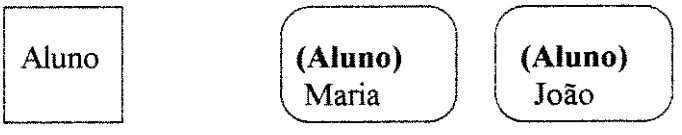
4.3 - Modelagem Estática

Neste tópico, é enfocada a representação gráfica de classes e objetos, e seus relacionamentos e estruturas internas.

4.3.1 - Classes e Objetos

A primeira notação que deve ser introduzida consiste na diferenciação de classes e de objetos (instanciações de classe).

Na figura 4.2 (a) pode ser observado a representação de uma classe *Aluno*, enquanto na (b) tem-se dois objetos *Maria* e *João* (instanciações da classe *Aluno*).



(a) (b)

Figura 4.2 - Representação inicial de classe e objetos (a) e (b)

Nota-se, pela figura acima, que a diferenciação entre a classe e suas instanciações (objetos) dá-se pela curvatura da caixa na qual está contido o nome de identificação. As classes são representadas por cantos não arredondados, enquanto os objetos, além dos cantos arredondados contém, entre parênteses e destacado, o nome da classe originária.

4.3.2 - Atributos

Atributos são dados (variáveis) que caracterizam um objeto, podendo os mesmos sofrer alterações devido a ações ocorridas sobre os objetos. Os atributos

de um objeto são listados em uma área localizada abaixo da destinada ao nome da classe base, conforme é mostrado na figura 4.3.

Suponha-se que para caracterizar um aluno, são necessárias as seguintes informações: registro acadêmico, nome e curso. A introdução desses campos na classe *Aluno* é ilustrada na figura 4.3 (a) e que em (b) ilustra-se a instanciação destes campos.

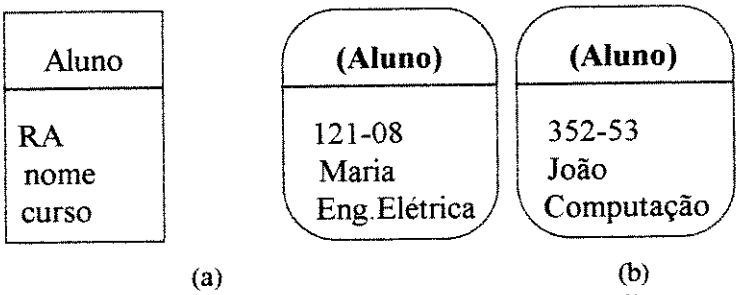


Figura 4.3 - Representação de classe e objetos com atributos

Os nomes dos atributos não necessitam ser mencionados nas instanciações dos objetos, apenas deverá ser respeitada a ordem de aparição em relação à classe base. Por tal motivo, nota-se que na figura 4.3 (b) são expressos apenas os valores dos atributos correspondentes à classe da figura 4.3 (a).

4.3.3 - Operações e Métodos

Operações são ações aplicadas sobre um objeto de uma classe, modificando seu estado. Cada operação é identificada, além de seu nome, pela sua assinatura, representada pelo número e tipos dos argumentos, assim como o tipo de retorno da operação.

A implementação das operações é denominada de método e pode ser acessada por objetos do mesmo tipo ou por outros que possuam permissão de acesso. Na figura 4.4 é mostrada a representação de uma classe completa: identificador, atributos e operações.

Em relação à figura 4.4, no primeiro exemplo, a operação *ReqHistóricoEscolar* não necessita de parâmetros; no segundo exemplo (*Req.Matrícula*) é passado como parâmetro o nome da matéria na qual está sendo solicitada a matrícula; no último exemplo, há a possibilidade de o aluno verificar se foi atendido na solicitação de matrícula na matéria *nome*, sendo que o resultado é coletado como retorno da função, *Boolean* neste caso.

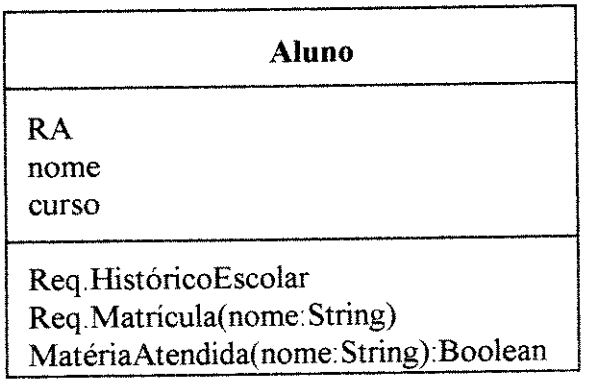


Figura 4.4 - Exemplo de uma classe completa

4.3.4 - *Links* e Associações

A conectividade entre objetos é conseguida por intermédio de *links*, interligando dois ou mais objetos de forma a relacioná-los mediante uma regra de associação. A figura 4.5 ilustra um relacionamento entre as classes *Universidade* e *Alunos*: a *Universidade* possui um *Reitor* e, por conseqüente, um *Reitor* administra uma *Universidade*.

Na figura 4.5, a nomenclatura da associação entre os dois tipos de classes foi atribuída levando-se em conta a expressão: uma *Universidade* possui um *Reitor*.

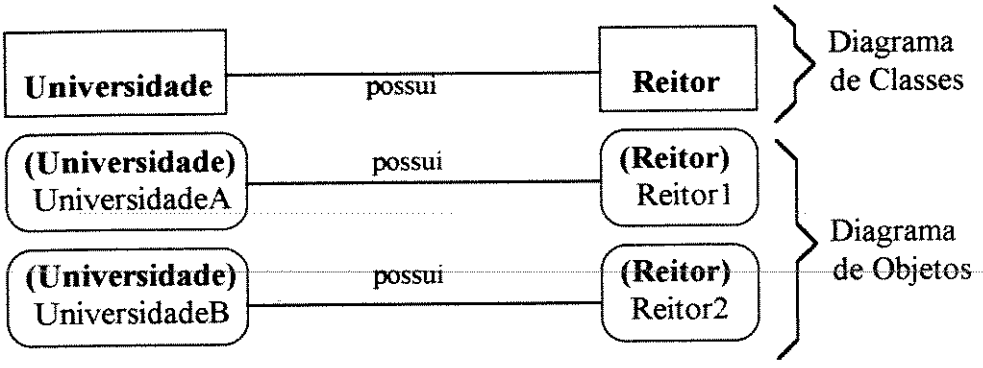


Figura 4.5 - Associação entre as entidades *Universidade* e *Reitor*

4.3.5 - Cardinalidade

A associação vista até então apresenta uma cardinalidade de 1:1. O modelo de representação de objetos manipula relações diferentes das unárias. Para representação da multiplicidade, atribui-se aos objetos correlacionados valores ou intervalos correspondente ao número de possíveis instanciações.

A figura 4.6 mostra diversas situações de cardinalidade envolvendo o relacionamento entre as classes *A* e *B*. Na figura nota-se o uso de dois símbolos relacionais: a circunferência preenchida, que identifica que poderão envolver muitas relações entre as referidas classes (zero ou mais). No item (a), não há a especificação de limite, representando que um objeto da classe *A* poderá interagir com zero ou mais objetos da classe *B*. A multiplicidade pode ser expressa em termos de valores mínimos, como o exemplo em (b): objetos *A* relacionam-se com um ou mais objetos *A*; em termos de faixa de valores (c); valores distintos (d); valores exatos (e): objetos da classe *A* interagem com exatamente três objetos da classe *A*. Por outro lado, pode-se optar pela associação entre zero ou um objeto (f), representada pela circunferência não preenchida: classe *A* relaciona-se com nenhum ou exatamente um objeto da classe *B*.

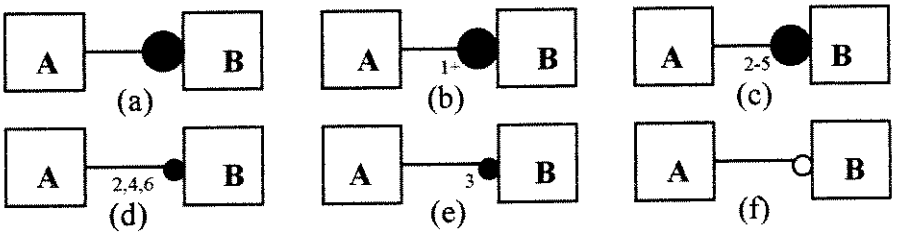


Figura 4.6 - Associações *n-árias* entre classes

4.3.6 - Atributos de Associação

Para uma melhor identificação na associação entre objetos é permitido, na metodologia OMT, inserir-se atributos nos *links*. Dessa forma consegue abstrair-se as associações ocorridas no mundo real.

A figura 4.7 mostra uma associação, entre as classes *Residência* e *Família*, englobando o atributo *endereço*.

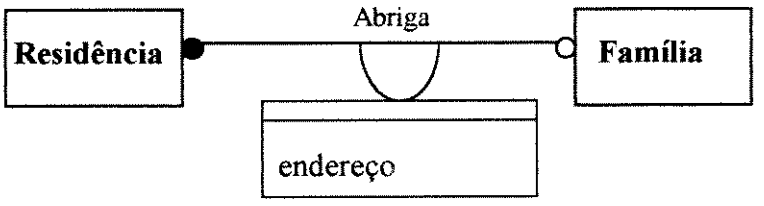


Figura 4.7 - Inserção de atributos no relacionamento entre *Residência* e *Família*

Na ilustração acima, pode-se observar que este relacionamento indica: "em uma residência mora uma ou nenhuma família e uma família pode possuir nenhuma ou várias residências". Existe ainda um atributo que identifica os endereços nos quais uma família reside.

4.3.7 - Denominação dos Papéis

Em uma associação, os papéis dos objetos envolvidos podem ser identificados pela inclusão dos termos contratuais associados a cada um dos mesmos.

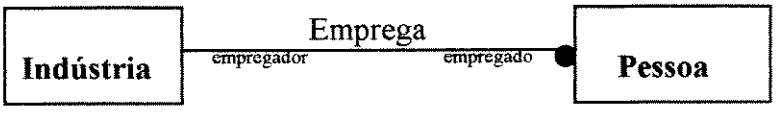


Figura 4.8 - Papéis desempenhados pelas classes *Indústria* e *Pessoa*

A figura 4.8 mostra uma associação entre os objetos da classe *Indústria* e *Pessoa*, explicitando o nome dos papéis desempenhados por estes.

4.3.8 - Agregação

Agregação diz respeito às associações entre objetos com o intuito de especificar os componentes formadores destes. A representação da agregação é realizada por intermédio de um losango interligando as classes base com as derivadas conforme é ilustrado na figura 4.9.

A referida figura diz respeito ao conceito de agregação, onde as associações verticais expressam as sentenças *"é formado por"* e *"é parte de"*. Tais expressões são utilizadas para obter o relacionamento descendente (da classe base para as derivadas) ou ascendente (das classes derivadas para a base), respectivamente.

Dessa forma, ainda na figura 4.9, é identificado que um automóvel *é formado pelas* partes mecânica, elétrica, de lataria e estofamento e, por sua vez, a parte mecânica é formada pela suspenção e motor e assim por diante. Por outro

lado, se a visão partir das classes derivadas, pode-se obter: as partes mecânica, elétrica, lataria e estofamento *são partes de* um automóvel.

Convém salientar que no exemplo exposto, as partes que integram fielmente um automóvel não foram cobertas na íntegra.

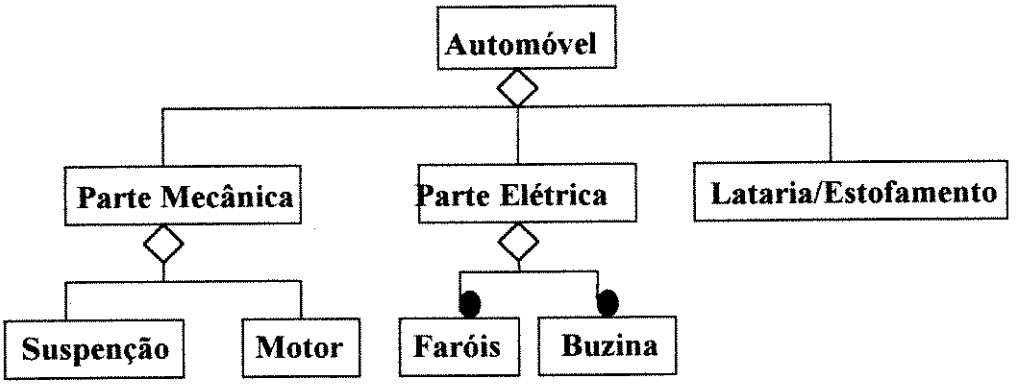


Figura 4.9 - Agregação entre os componentes de um automóvel

São reconhecíveis, a partir do conceito de agregação, três propriedades fundamentais:

- transitividade: se *A* é parte de *B* e, por sua vez, *B* é parte de *C*, então *A* é parte de *C*.
- antissimetria: se *A* é parte de *B*, pode-se afirmar que *B* não é parte de *A*.
- propagação: supõe-se uma classe *A* formada pelas classes *B* e *C*. Caso ocorram alterações em algum dos componentes (*B* e/ou *C*), por conseqüência, as alterações são refletidas por *A*.

4.3.9 - Generalização/Especialização

Conforme já relatado no item 4.2.5, os conceitos de generalização e de especialização estão intimamente relacionados com o de herança. O modelo OMT distingue a herança simples, a qual uma classe descende de apenas uma outra, da herança múltipla, com a possibilidade de uma classe ser originária de duas ou mais.

A representação de uma herança simples é feita por intermédio de um triângulo, relacionando a classe base e suas classes derivadas, e pode ser observada na figura 4.10. Na citada figura, na especialização da classe *Veículo* originam-se as classes derivadas *Veículo Terrestre*, *Veículo Aquático* e *Veículo Aéreo*. Por sua vez, um caso de herança múltipla ocorre na classe *Veículo Anfíbio*, herdando as características das classes *Veículos Terrestres* e *Veículos Aquáticos*.

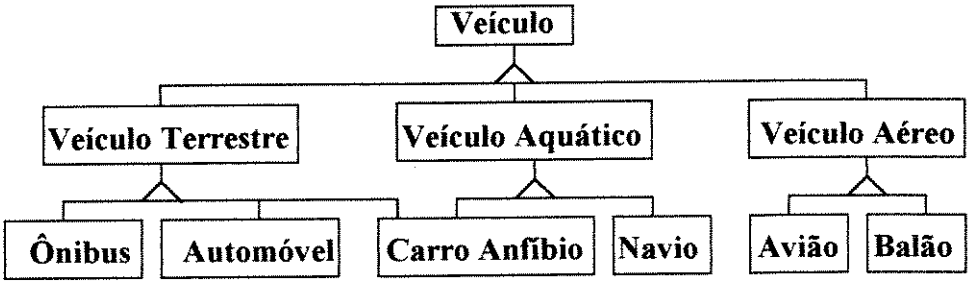


Figura 4.10 - Representação de Associações Hierárquicas

4.4 - Modelagem Dinâmica

Diferentemente da modelagem estática, a modelagem dinâmica incorpora a ação temporal em sua representação, descrevendo os objetos por intermédio dos eventos e estados assumidos por eles.

Eventos são ações aplicadas sobre objetos, possuindo, teoricamente, um tempo de ocorrência igual a zero. Pela atuação dos eventos, objetos podem sofrer transição em seu estado atual. Define-se estado como sendo os valores dos atributos e o conjunto de *links* mapeados por um objeto em um dado instante.

A representação dinâmica da modelagem por objetos envolve três maneiras distintas: diagrama de transição de estados, histórico de eventos e cenários.

Para melhor enfocar a representação dinâmica, os modelos de modelagem dinâmica abordados abaixo têm como exemplo o processo de criação de um programa partindo-se da modelagem do código fonte. No mesmo, é considerado um arquivo-fonte em suas etapas de confecção, compilação, linkagem e o próprio código executável.

4.4.1 - Diagrama de Estados

Mapeia a transição de estados dos objetos motivada pela ativação de certos métodos. O novo estado, portanto, depende de dois fatores: o estado no qual se encontra o objeto no momento da ocorrência do evento e o próprio evento disparado.

A representação do diagrama é feita mediante arcos e nós. Cada nó representa um possível estado do objeto, interligados pelos arcos que expressam os eventos aplicáveis sobre cada estado.

Na figura 4.11 são representados os estados ocorridos desde a fase de implementação de um programa até a geração final do código executável e as operações responsáveis pela transição de tais estados.

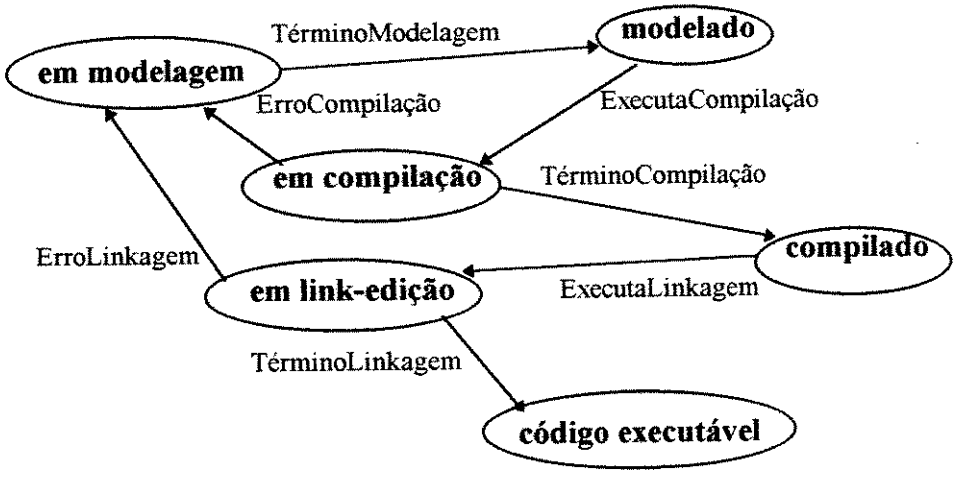


Figura 4.11 - Etapas no processo de implementação de um programa

Na figura acima, os estados são representados pelas elipses e as transições pelos arcos com os nomes das operações correspondentes.

4.4.2 - Histórico de Eventos

O histórico de eventos tem como objetivo explicitar o intercâmbio de mensagens entre objetos ou seus estados, envolvendo as características temporais para representá-lo.

A modelagem dinâmica mediante o histórico de eventos é realizada particularmente para cada meta estipulada. A figura 4.12 ilustra as operações envolvidas tendo como meta a compilação de um programa, representando portanto os estados *em modelagem*, *modelado*, *em compilação* e *compilado*.

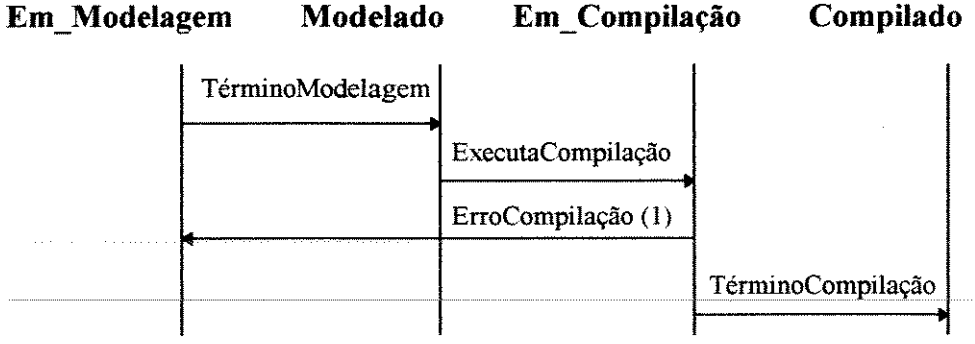


Figura 4.12 - Operações envolvidas durante o processo de compilação

Ainda na figura 4.12, a operação marcada por (1) somente é executada na ocorrência de algum erro de compilação, voltando ao estado "*Em Modelagem*", devendo-se neste caso, reiniciar todo o ciclo. Na ausência de erro, a operação (1) inexiste, passando então imediatamente para a operação "*TérminoCompilação*".

Devido à possibilidade de retorno ao estado inicial (*em modelagem*), o diagrama seria corretamente colocado sob dois aspectos: com e sem erro de compilação conforme ilustra a figura 4.13a e 4.13b, respectivamente.

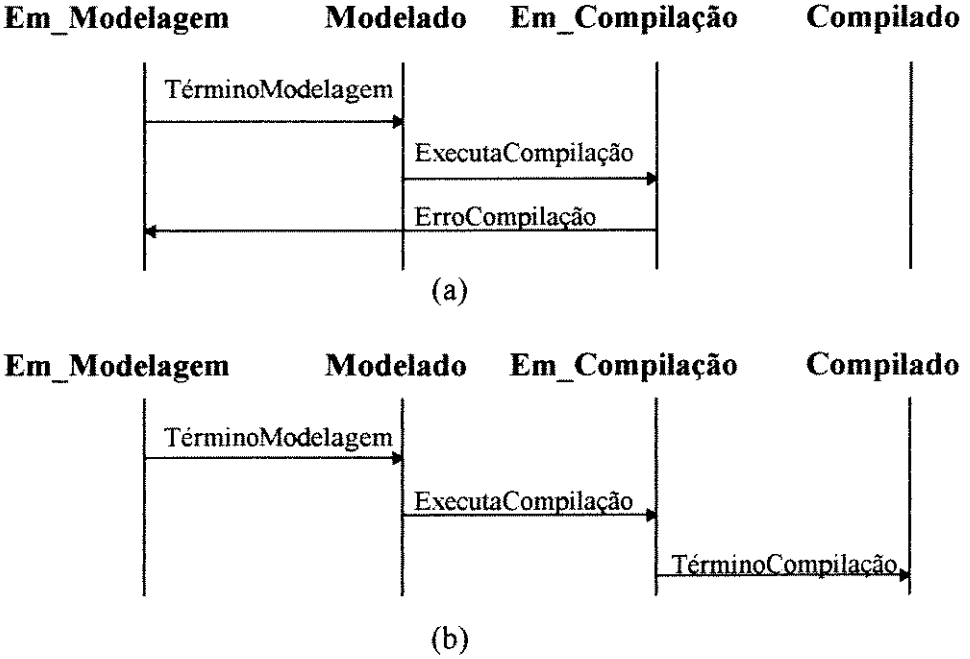


Figura 4.13 - Processo de Compilação com e sem a Ocorrência de erro (a) e (b)

4.4.3 - Cenários

Os cenários consistem em textos descritivos abrangendo as operações envolvidas para a realização de um certo objetivo.

No exemplo utilizado, um possível cenário poderia ser elaborado por intermédio do seguinte texto:

> *Etapas para obtenção de um código executável a partir de um código-fonte:*
>
> *Define-se como código-fonte a seqüência de instruções escritas pela utilização de uma linguagem de programação armazenada em um arquivo. Define-se ainda como código-executável o arquivo contendo as instruções em linguagem de máquina.*
>
> *Após completada a modelagem, um arquivo-fonte poderá sofrer uma etapa de compilação para que seja obtido um arquivo contendo o código compilado, caso ocorra algum erro em tal etapa, o arquivo deverá sofrer um reformulamento de modo a corrigir os erros listados.*
>
> *Uma segunda etapa a seguir consiste na link-edição do arquivo compilado de modo a obter, como resultado, o código executável. Assim como a etapa de compilação, ocorrendo alguma falha durante a etapa de linkagem, o arquivo-fonte deverá ser modificado para que os erros sejam sanados.*

4.5 - Modelagem Funcional

Modelagem funcional descreve o processamento dentro de um sistema computacional. Este modelo enfoca como os valores na saída de um módulo de processamento são derivados daqueles entregues à sua entrada.

A descrição dos módulos de gerenciamento é feita somente sob a forma funcional, abordando-os como *black-box*, estabelecendo-se a relação entre os dados presentes na sua entrada e aqueles presentes na sua saída (figura 4.14).

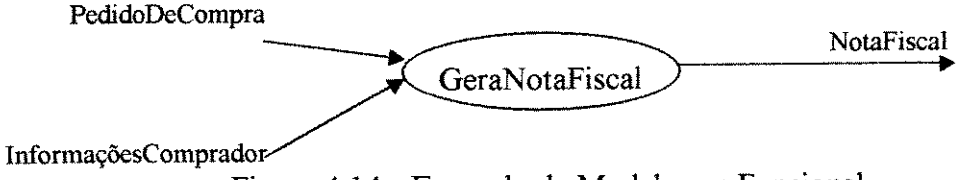


Figura 4.14 - Exemplo de Modelagem Funcional

Na figura 4.14 são representados um módulo de processamento (*GeraNotaFiscal*) e três fluxos de dados (*PedidoDeCompra*, *InformaçõesComprador* e *NotaFiscal*). Nota-se que a única preocupação da modelagem funcional é estabeler os fluxos de informações envolvidos com os módulos de processamento, não havendo o enfoque em relação ao controle do

processamento ou a estrutura computacional presente no módulo de processamento.

Diversas técnicas podem ser utilizadas para modelar o sistema sob o aspecto funcional, dentre as quais destaca-se o diagrama de fluxo de informações [Page-Jones88].

Capítulo 5 - Especificação da Camada Groupware

5.1 - Introdução

Atualmente, observa-se uma grande preocupação em relação ao contexto de ambientes heterogêneos e à cooperação de usuários sob uma meta comum, aliada à crescente manipulação de informações multimídia.

A *Plataforma Multiware* foi projetada com o intuito de fornecer mecanismos que abrangem tanto os aspectos de ambientes distribuídos quanto os aspectos de suporte às atividades de grupo.

Dentre as camadas internas à plataforma, será enfocada a *Groupware*, responsável por suprir serviços de suporte a grupos cooperativos.

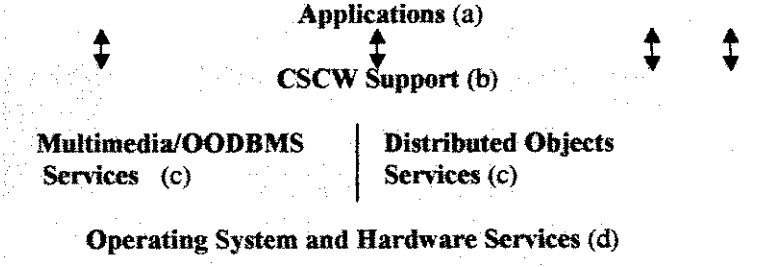
Este capítulo foi dividido nas seguintes partes: primeiramente encontra-se a descrição da Plataforma Multiware, caracterizando-se as camadas internas e, na seqüência são listados os serviços oferecidos pela Camada Groupware.

5.2 - Plataforma Multiware

A Plataforma Multiware integra serviços de computação distribuída, manipulação multimídia e suporte a grupos para que sejam utilizados em diversas aplicações [Mendes93; Loyolla94b].

Possui como meta principal suportar a nova geração de aplicações, as quais requerem um ambiente computacional distribuído e heterogêneo. Dentre tais aplicações destacam-se aquelas envolvidas com Trabalho Cooperativo (CSCW), Inteligência Artificial Decentralizada (DAI) e Sistemas de Suporte a Tomada de Decisões.

A Camada Multiware é formada por três camadas internas: Groupware Layer, Middleware Layer e Low-Level Layer, conforme a representação contida na figura 5.1.



(a) Final Users (b) Groupware Layer (c) Middleware Layer (d) Low-Level Layer

Figura 5.1 - Estruturação da Plataforma *Multiware*

A escolha pela utilização do Modelo de Referência para Processamento Aberto e Distribuído (RM-ODP) e do CORBA (*Common Object Request Broker Architecture*), este como implementação comercial de plataforma distribuída,

possibilitou a garantia de transparência de manipulação frente ao usuário e a heterogeneidade entre ambientes computacionais distintos.

5.2.1 - Low-Level Layer

Utilizada para interfacear a plataforma Multiware com o ambiente nativo (*hardware* e sistema operacional) sobre o qual está instalada. Nele encontram-se os elementos básicos do sistema operacional e todo o suporte para a sincronização de processos, protocolos de comunicação, mecanismos de acesso a arquivos, dentre outros.

5.2.2 - Middleware Layer

A camada *Middleware* é responsável por fornecer a infra-estrutura para o processamento distribuído aberto e de informações multimídia para a camada *Groupware* e para aplicações. O emprego da camada *Middleware* tem como principal vantagem a independência das aplicações em relação ao ambiente nativo do sistema computacional [Loyolla94b].

Tais funcionalidades são desempenhadas por duas sub-camadas: *Multimedia Processing Sublayer* e *ODP Sublayer*.

5.2.2.1 - Multimedia Processing Sublayer

A sub-camada de processamento multimídia é responsável pelas seguintes funções:

- Transporte de áudio e vídeo pela rede de comunicação
- Suporte de tempo-real para objetos multimídia
- *Scheduling* em tempo-real para objetos multimídia
- Sincronização de objetos multimídia
- Apresentação de informações multimídia
- Armazenamento e recuperação de objetos multimídia

O transporte de áudio e vídeo pela rede de comunicação foi implementado sobre o *Internet Protocol* (IP) suportando dois tipos de mecanismos: "reliable multicast" de mensagens e "fast delivery" de informações de áudio e vídeo digitalizadas.

A comunicação entre os objetos multi-mídia é realizada seguindo os padrões propostos pelo RM-ODP, envolvendo os conceitos de *binding* e canais, enquanto que o escalonamento é conseguido pela utilização de *threads* e a sincronização pelo método *Logical Time System* [Anderson91].

A unidade de escalonamento (*scheduling*) consiste em um *thread*, o qual tipicamente é utilizado sobre a implementação de interfaces dos objetos multimídia. Por sua vez, a unidade para apresentação das informações multimídia foi implementada utilizando as funcionalidades de interfaces comerciais, tais como o *Presentation Manager* [Tizzo95].

O armazenamento e recuperação dos objetos multimídia é realizado por intermédio de uma base de dados distribuída orientada para objetos [Tobar94; Tobar95; Ricarte96].

5.2.2.2 - ODP Sublayer

Para incorporar os conceitos definidos pelo RM-ODP e os mecanismos já fornecidos pelos sistemas distribuídos atualmente disponíveis comercialmente, a sub-camada ODP foi dividida em níveis conforme ilustra a figura 5.2 [Maezi95; Gunji95; Araújo96].

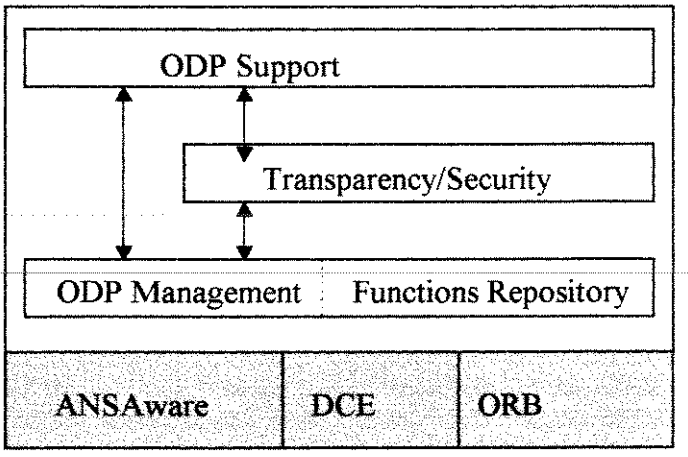


Figura 5.2 - Divisão Estrutural da sub-camada ODP

O motivo desta divisão reside no fato que as plataformas ditribuídas disponíveis comercialmente (nível inferior) não suportam, em sua totalidade, o desenvolvimento, uso e manutenção das aplicações distribuídas, sendo necessário a incorporação de código adicional (nível superior) para que a camada venha a suportar os conceitos de sistemas distribuídos abertos.

Por sua vez, o sub-nível *ODP Support* apresenta-se composto de quatro partes: *Applications Support*, *Groups Support*, *Trading*, *Transactions Support*, conforme é ilustrado na figura 5.3.

Figura 5.3 - Divisão Estrutural do sub-nível *ODP-Support*

Os sub-níveis da sub-camada *ODP* desempenham as seguintes funções:

- *Applications Support*: engloba funcionalidades para a definição e a instanciação de objetos, assim como mecanismos de suporte a transparência, segurança e comunicação entre os mesmos (binding).
- *Trading*: exporta serviços de negociação entre objetos servidores e objetos clientes.
- *Groups Support*: fornece facilidades para o suporte das atividades que envolvem grupos de trabalho.

- *Transactions Support*: fornece suporte às operações de transação entre objetos.

5.2.3 - Groupware Layer

A camada *Groupware* visa fornecer suporte aos serviços cooperativos em termos de criação, gerenciamento, distribuição de mensagens e fechamento dos mesmos. Para tanto, a camada exportará, por intermédio de uma plataforma de objetos distribuídos, métodos aos requisitantes (usuários), parametrizados segundo suas interfaces.

A camada se distribuirá dinamicamente ao longo do ambiente computacional, porém manterá um objeto permanentemente instanciado e centralizado, o qual desempenhará a função de negociar o contato inicial entre os membros e a camada cooperativa [Loyolla94a].

Um dos requisitos básicos de uma camada cooperativa é garantir a autonomia dos membros participantes associados aos serviços cooperativos gerenciados por tal plataforma. A autonomia é garantida pela cobertura de certos quesitos, tais como sincronismo, segurança, qualidade operacional, transparência e o próprio gerenciamento de sistemas distribuídos [Bannon91].

5.3 -Serviços Oferecidos pela Camada Groupware

Com o objetivo de fornecer mecanismos de suporte à cooperação, as funcionalidades exportáveis pela camada *Groupware* podem ser classificadas segundo o tipo de atuação dos usuários frente aos grupos cooperativos: serviços de contactação e serviços de manutenção.

Fazem parte dos serviços de contactação aqueles necessários para a criação e encerramento de grupos, anexação de novos membros, requisição de informações de grupos já criados, assim como a relação dos membros já integrados pelos mesmos. Por sua vez, os serviços de manutenção são responsáveis pela interação entre os membros integrantes de um certo grupo e pela configuração do ambiente cooperativo face às qualidades de serviço locais e globais envolvidas no serviço.

A figura 5.4 representa os serviços exportados pela camada *Groupware* em relação ao tipo de usuário, o qual pode ser interno ou externo a um grupo.

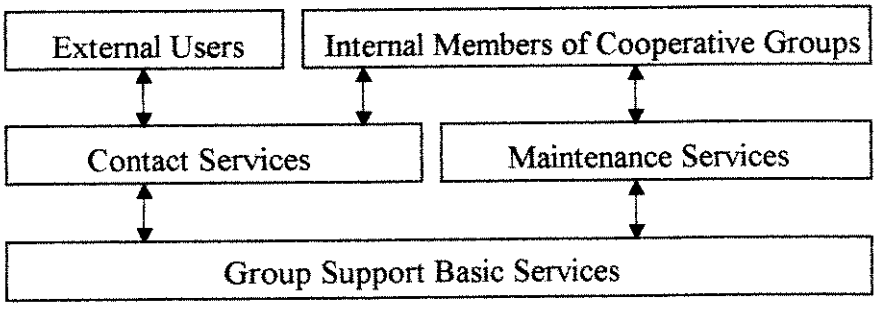


Figura 5.4 - Visão dos Serviços Exportados pela Camada Frente aos Usuários

Uma melhor descrição de tais métodos se encontra no capítulo referente aos aspectos de implementação (capítulo 7), dentro das explanações dos objetos integrantes à camada *Groupware*.

5.3.1 - Serviços de Contactação

Os serviços de contactação têm como objetivo estabelecer um elo de ligação entre os membros, envolvidos ou não a um grupo cooperativo. Os serviços de contactação são necessários para que sejam realizadas as etapas de criação, iniciação das ações cooperativas e encerramento de um grupo, assim como obter informações sobre aqueles já criados e requisitar a introdução de um membro ainda não pertencente a um grupo já em andamento.

Subdividem-se em serviços de criação, encerramento e ativação de grupos, adição de novos membros a grupos em andamento e dos serviços para ativação das ações cooperativas locais.

5.3.1.1 - Serviços de Criação, Encerramento e Ativação de Grupos Cooperativos

- *ContructNewGroup*: Responsável pela criação de um novo grupo cooperativo. Em primeira instância, para a construção de um novo grupo, informações pertinentes ao mesmo deverão ser apresentadas. Tais informações constam de: tipo do serviço cooperativo, nome, comentário, momento para o início das atividades, duração do cooperativismo, tipo do objeto gerenciador de membros, relação dos membros que farão parte do grupo, número de membros presentes na relação, descrição dos recursos que serão alocados, políticas de construção, gerenciamento e término, número de políticas presentes na lista.

- *InitGroup*: Objetiva a ativação de um Grupo já estruturado. Em termos de condições funcionais, um grupo cooperativo pode se encontar em um dos seguintes estados: em confecção, pronto para a execução, em execução ou interrompido. A transição do estado "pronto para execução" para "em execução" pode realizar-se de dois diferentes modos: automática ou manual. No modo automático, o momento para o início das ações cooperativas é indicado no momento da criação do grupo, enquanto que no modo manual, o início será efetuado diretamente pelo membro que requisitou a criação do grupo.

- *CloseGroup*: Quando evocado, processa todas as etapas necessárias para o encerramento das atividades de um grupo cooperativo.

5.3.1.2 - Serviços de Adição de Novos Membros a Grupos já em andamento

- *AddNewMember*: Pedido de inclusão de um membro a um grupo já em andamento. Para adicionar-se um membro durante a evolução das atividades coopeartivas de um grupo, a aplicação deve permitir a entrada de todas as informações relativas ao mesmo, assim como as dos recursos que estarão à ele associados.

5.3.1.3 - Serviços de Ativação das Ações Cooperativas Locais

- *InitMember*: Iniciação da ação cooperativa local. No momento do início das ações cooperativas de um grupo, os membros integrantes poderão estar desconectados ao grupo, incorporando-se ao longo da evolução das atividades do mesmo. A etapa de conexão representa o início das ações cooperativas locais, onde a camada envia uma mensagem a todos os demais membros conectados, informando que o recém-conectado membro iniciou as suas atividades no grupo.

5.3.1.4 - Serviços de Informações sobre grupos

- *GetCreatedInfo*: Fornece informações sobre os grupos cooperativos criados. As informações consistem de: número de grupos criados, nome dos grupos criados, sua localização em relação aos *hosts* e o nome do membro responsável pelo grupo.

5.3.2 - Serviços de Manutenção

Os serviços de manutenção são exportados pela Camada Groupware para forncecer condições necessárias para a cooperação entre os membros de um certo grupo cooperativo. Subdividem-se em serviços de manipulação dos canais de informação, envio de mensagens, serviços de questionamento e de saída ou interrupção de membros frente ao grupo cooperativo.

5.3.2.1 - Serviços de Manipulação dos Canais de Informação

- *OpenInformalChannel*: Realiza a abertura de um canal informal de comunicação. Canal informal representa um meio de comunicação temporário interligando dois membros de um mesmo grupo para que possam trocar mensagens paralelamente ao andamento do grupo. O canal informal baseia-se no modo de operação *unicast*, envolvendo apenas dois membros: um transmissor e outro receptor.

- *CloseInformalChannel*: Responsável pelo fechamento do canal informal de comunicação.

- *ChangeCurrentMember*: Quando evocado, o serviço realiza a mudança do membro correntemente detentor da utilização do canal formal de comunicação. Entende-se como canal formal o meio de comunicação que interliga um membro (membro corrente) aos demais envolvidos com o grupo. Tal canal opera por meio de *broadcast*, onde a informação proveniente do membro corrente é distribuída para todos os outros membros. Na evolução das ações cooperativas, é permitida a rotatividade

dos membros correntes, havendo a notificação da mudança a todos os membros realizada automaticamente pela Camada Groupware, através do canal de informações gerais.

5.3.2.2 - Serviços de Envio de Mensagens

- *SendInformalMessage*: Permite enviar mensagens informais por um canal informal já estabelecido.

- *SendFormalMessage*: Este serviço poderá apenas ser acessado pelo membro detentor do direito de utilizar o canal formal de comunicação para que possa enviar as mensagens à todos os demais membros ativos do grupo cooperativo.
- *SendGeneralMessage*: Serviço reservado para a aplicação ou para a Camada Groupware, de modo a possibilitar o envio de avisos relacionados ao andamento da ação cooperativa.

5.3.2.3 - Serviços de Questionamento (votação)

- *AnswerQuestion*: Quando a aplicação necessita de questionamentos, um membro poderá remeter suas respostas aos assuntos solicitados por intermédio da ativação deste serviço.

5.3.2.4 - Serviços de Saída/Interrupção do Membro Frente ao Grupo

- *InterruptMember*: Um membro pode se desligar temporiamente do grupo no qual está engajado mediante a evocação deste serviço. Quando evocado, a Camada Groupware enviará uma mensagem aos demais membros, pelo canal de mensagens gerais, informando o novo estado do membro recém desconectado.

Capítulo 6 - Projeto da Camada Groupware

6.1 - Introdução

Conforme descrito anteriormente, há uma necessidade da representação de estado da cooperação segundo critérios temporais e sociais para que haja um perfeito gerenciamento *isócrono* ou *anisócrono* da aplicação cooperativa.

Um modelo representativo simbolizado pelos relacionamentos de heranças e associações entre os objetos da camada *Groupware*, permite que seja alcançado um contexto distributivo por entre os *hosts* participantes do serviço. Tal distribuição permite um melhor equilíbrio de cargas e um grau maior de abstração.

Neste capítulo serão abordados os detalhes relevantes do projeto da camada *Groupware*, enfocando o interfaceamento entre os seus objetos.

6.2 - Sub-Camadas da Camada *Groupware*

Sob ponto de vista funcional, a camada *Groupware* apresenta sub-camadas distintas, cuja função é proporcionar um gerenciamento menos complexo e mais eficiente das informações de dados e de controles que trafegam pela camada. A figura 6.1 ilustra a estruturação da Camada Groupware segundo a funcionalidade das sub-camadas.

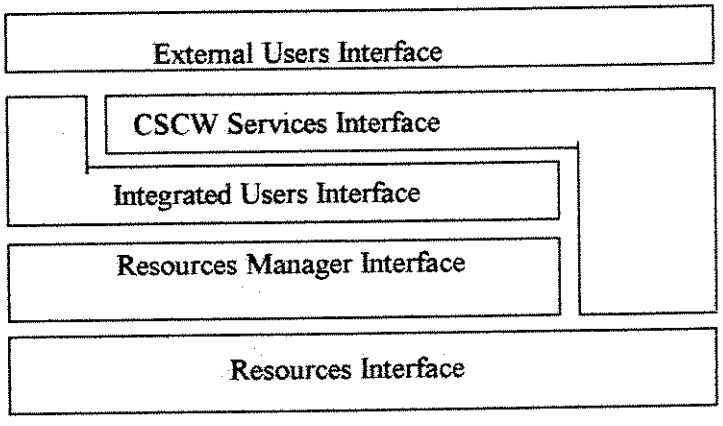


Figura 6.1 - Distribuição Funcional das Sub-Camadas Cooperativas

Tendo um enfoque totalmente cooperativo, a camada não se limita apenas ao tratamento das requisições dos membros já agregados, mas abre a possibilidade de requisições de membros externos ao grupo, tratadas pela interface representada pela sub-camada *External Users Interface*. Tal camada visa interfacear com o meio externo e rotear mensagens para os grupos específicos, além de realizar a efetiva construção de um novo grupo cooperativo.

A sub-camada *CSCW Services Interface* abriga objetos que representam os grupos cooperativos criados, possuindo a função de gerenciar o grupo representado e manter a qualidade de serviço em condições operacionais.

Os mecanismos de cooperação entre os membros integrantes de um grupo cooperativo são fornececidos pela sub-camada *Integrated Users Interface*. Além de englobar os objetos representativos dos usuários, referencia também um objeto cuja função é de gerenciar o grupo frente à troca de mensagens entre os membros.

O mesmo gerenciamento descrito anteriormente se aplica a nível de recursos, estabelecendo-se uma política de qualidade e prioridades. Uma classe de objetos embutida na sub-camada *Resources Manager Interface* supre o ambiente de tais serviços.

A imensa disponibilidade de recursos passíveis de serem utilizados por ambientes cooperativos colaborou para o surgimento da sub-camada de mais baixo nível aqui referenciada: trata-se do *Resources Interface*, que faz a interação do ambiente com os próprios recursos alocados. Dentre os tipos recursos, também se encontram os adaptadores para *OODBMS* e canais do tipo *stream*.

A comunicação entre a camada e os membros e estes entre si é realizada por intermédio de canais de comunicação. Dentro do comportamento de grupos, as mensagens distingüem-se em relação ao seu conteúdo, ao seu emissor e ao seu destinatário. Desta forma, foram implementados três tipos de canais de comunicação:

- *canais formais de comunicação*: feita a partir de um membro detentor do direito de voz, enviando suas mensagens a todos os demais membros do grupo

- *canais informais de comunicação*: utilizados para a comunicação informal, abertos dinâmica e aleatoriamente entre dois participantes quaisquer

- *canais de mensagens gerais*: mensagens da própria camada, ou da aplicação cooperativa, para os membros ativos do grupo sob gerenciamento

6.3 - Localização das Referências dos Objetos da Camada Groupware

Assumindo-se uma abordagem distribuída, há a necessidade de uma referenciação em relação à localização dos objetos. Para tanto, a referência dos objetos será classificada como: referências de objetos fixas e centralizadas, referência de objetos centralizadas no *chairman* e referência de objetos locais.

As referências de objetos fixas e centralizadas são aquelas que serão utilizados como interfaceamento entre os membros da coletividade (externos) e os próprios serviços cooperativos, não podendo sofrer migrações para que sejam facilitados os processos de localização e de acesso. Pode-se dizer, então, que os objetos referenciados como fixo e centralizados representam um servidor de serviços cooperativos.

De forma semelhante, existem as referências de objetos centralizadas no *chairman*, as quais são responsáveis pela manutenção do serviço cooperativo corrente e pelo interfaceamento dos recursos comuns ao grupo.

Por último, existem as referências dos objetos locais, simbolizando o tratamento e gerenciamento dos recursos locais a cada membro participante.

Pode-se correlacionar a distribuição dos objetos com uma hierarquia de camadas funcionais, onde o mais alto nível representa a própria aplicação e os membros externos; o nível mais baixo, integra os membros e os recursos alocados cooperativamente e, por último, a camada intermediária incorpora a tarefa de gerenciamento em relação ao grupo como um todo. O primeiro nível serve como porta de comunicação entre os membros externos e o núcleo de gerenciamento da Camada Groupware. Os mecanismos de interação entre os membros e os recursos do grupo competem ao nível mais baixo.

A distribuição hierárquica dos tipos de referências dos objetos pertencentes à Camada Groupware é representada na figura 6.2.

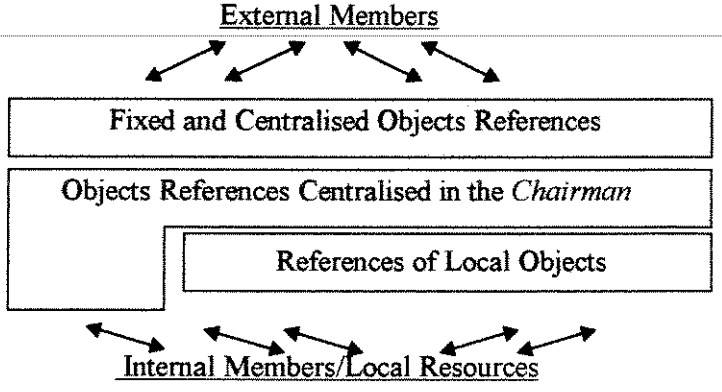


Figura 6.2 - Hierarquia dos Objetos da Camada Groupware

6.4 - Objetos Integrantes da Camada Groupware

Após o reconhecimento das funções (serviços) básicas a serem cobertas por cada sub-camada, mencionadas no capítulo 5, torna-se possível a total abstração a nível de representação por objetos de modo a garantir a atuação em ambientes distribuídos de forma isócrona ou anisócrona.

Um dos requisitos básicos de uma aplicação cooperativa, além dos já citados, é visar a garantia de gerenciamento de agentes (membros/recursos) dentro de um ou mais grupos ativos.

O modelo de objetos da Camada Groupware está representada na figura 6.3. Nesta figura é utilizada a notação por objetos proposta por Rumbaugh [Rumbaugh91].

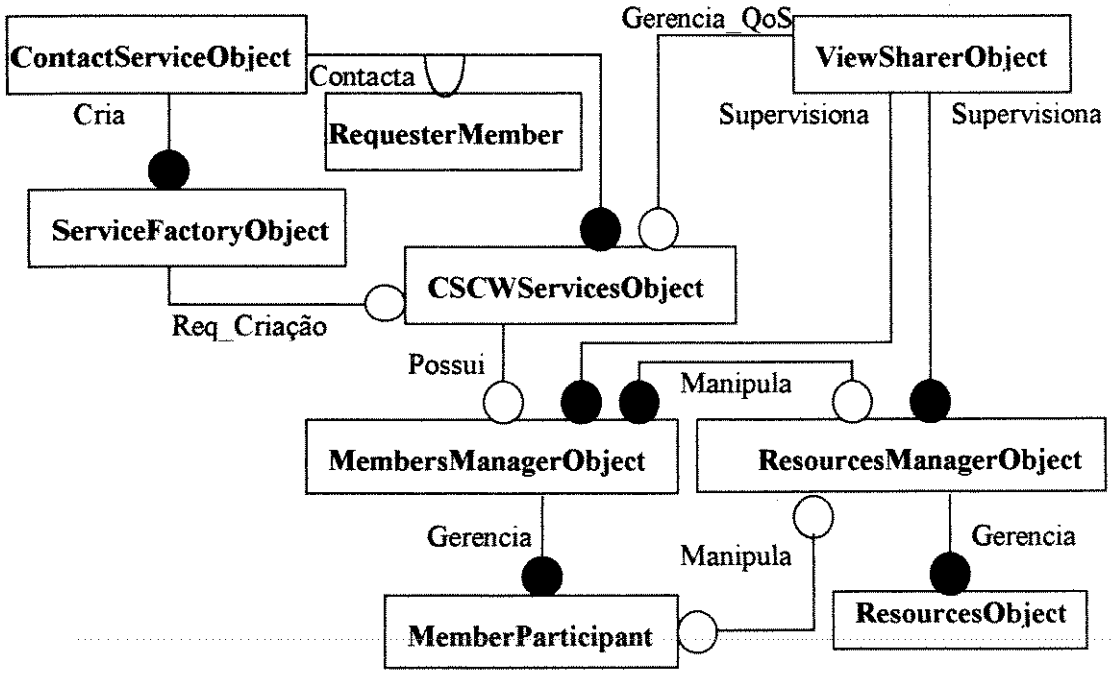


Figura 6.3 - Objetos Primários da Camada Groupware

6.4.1 - Objeto ContactServiceObject

Define-se comunidade de usuários como sendo o conjunto formado por todas as pessoas que têm possibilidade de algum tipo de conectividade com aplicações cooperativas, as quais integram o universo regido pela camada *Groupware*. Tais usuários são aqui definidos como membros da comunidade.

Há a necessidade de uma interação entre os membros da comunidade que não se encontram conectados a um serviço cooperativo em específico, para que possam efetuar diversas operações nas mesmas, tais como consultas, requisição de inclusão, pedidos de criação de novos serviços, etc. Para tanto, o *ContactServiceObject* foi projetado com tal intuito, suprindo amplamente as necessidades desses usuários externos.

O relacionamento de um membro externo à uma dada aplicação por intermédio do objeto *ContactServiceObject* é mostrado na figura 6.4.

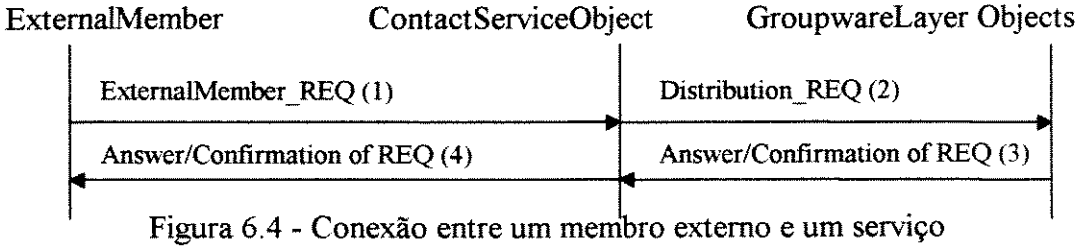


Figura 6.4 - Conexão entre um membro externo e um serviço

Nota-se, pela ilustração acima, que o *ContactServiceObject* irá propagar a mensagem do membro externo até o objeto responsável pelo controle ou pelo armazenamento da informação requerida, retornando a própria resposta à solicitação (um dado qualquer ou uma confirmação de um pedido específico). Por outro lado, se a informação/processo já estiver contido em tal objeto, haverá a omissão das etapas (2) e (3).

Nota-se que, por não realizar ações performativas, este objeto é considerado como sendo um *artefato*, pois apenas realiza um interfaceamento entre duas partes ou devolve o resultado de alguma pesquisa requisitada pelo membro da comunidade.

6.4.2 - Objeto ServiceFactoryObject

O objeto *ServiceFactoryObject* tem como objetivo tratar a etapa de criação de um serviço cooperativo. A sua instanciação, requisitada pelo *ContactServiceObject*, é necessária para a não sobrecarga deste, transferindo para si, a responsabilidade de confecção do grupo cooperativo. Pode realizar, quando a criação da aplicação é manifestada por convites, uma comunicação direta entre o membro criador do grupo (*chairman*) e os seus demais possíveis membros. Por outro lado, quando a participação é imposta, nenhuma consulta é realizada.

Para o estabelecimento do serviço, o requisitante (possível *chairman*) deverá informar:

- tipo de serviço a ser estabelecido
- nome do serviço
- breve comentário sobre o serviço
- possíveis membros que deverão compor o serviço e seus papéis (funções)

Informações adicionais deverão ser passadas ao objeto de criação para que possa manipular políticas de criação de grupos, tais como:

- Habilitar ou não a consulta aos possíveis membros, para atribuir-lhes a liberdade de escolha da participação ou não ao serviço para o qual foi convidado. A não habilitação desse item significa a imposição de participação dos membros listados pelo *chairman*. Possibilitando ainda que a introdução dos membros participantes seja de forma espontânea, deixando em aberto a relação das pessoas que integrarão, ou poderão integrar, um determinado serviço cooperativo.
- Outorgar ou não ao objeto a autoridade de decisão da criação do serviço dependendo das repostas dos membros consultados, estando o número de respostas afirmativas acima de um valor mínimo passado como parâmetro. A desconsideração dessa autoridade implicará na consulta ao membro requisitante, para que possa ele avaliar as respostas dos membros consultados (respostas coladas) e decidir a criação ou não do serviço.

Como resumo, o *ServiceFactoryObject* desempenha um papel de *administrador*, pois visa construir um novo grupo segundo uma certa política estabelecida para a criação. Em certas ocasiões, como na verificação do número de respostas positivas dos membros convidados, pode também atuar como um *árbitro*.

6.4.3 - Objeto CSCWServicesObject

O objeto *CSCWServicesObject* visa representar os próprios serviços cooperativos. Por possuir características comportamentais dinâmicas pode ser classificado como sendo um agente, pois se comporta de maneira diferenciada de acordo com as características do serviço em questão. Além de agente, também atua como um administrador, uma vez que será responsável por estabelecer uma certa política ao serviço de acordo com os parâmetros estabelecidos em sua configuração.

Tem a responsabilidade de instanciar, caso necessário, os objetos *MembersManagerObject* e *ViewSharerObject* (tópicos 6.4.4 e 6.4.5), este último para alguns tipos de serviços que necessitam de um acompanhamento permanente da qualidade de serviço.

Cada objeto do tipo *CSCWServicesObject* irá se conectar, no máximo, a apenas um *ViewSharerObject* e *MembersManagerObject*, pois haverá somente um grupo de usuários envolvidos com cada serviço cooperativo.

O relacionamento entre o *CSCWServicesObject* e os objetos *ContactServiceObject*, *MembersManagerObject* e *ViewSharerObject* é representado na figura 6.5.

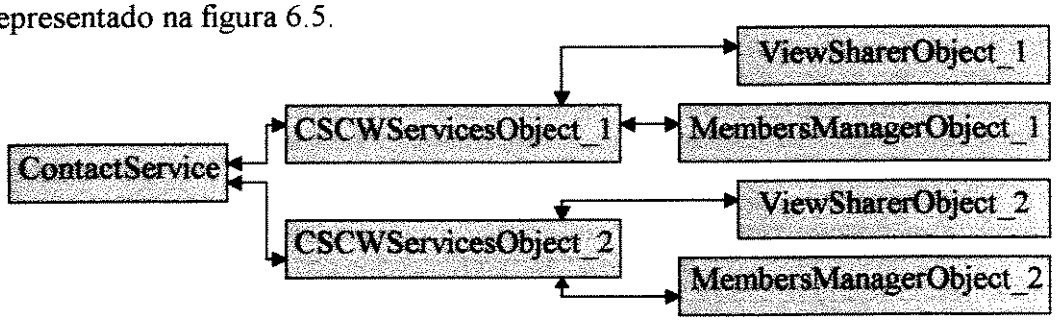


Figura 6.5 - Ligações a partir do *CSCWServicesObject*

Na figura acima, são ilustrados dois grupos cooperativos distintos sob um mesmo domínio. Nota-se que cada grupo opera independente através da instanciação de seus próprios objetos gerenciadores, interfaceadores e supervisores. Não foram mostrados, em tal figura, os demais objetos pertencentes à camada, os quais serão vistos oportunamente.

Um serviço cooperativo pode ser classificado quanto à sua forma de construção, ação temporal e controle da qualidade de serviço. Devido a esta classificação, o objeto *CSCWServicesObject* apresenta objetos especializados conforme são representados na figura 6.6.

Nota-se na figura 6.6 o aparecimento de especializações denotadas por *CSCWFullConstruct* e *CSCWParcialConstruct*. Como exemplo de um serviço com construção parcial, pode-se citar sistema de votação onde somente existem membros externos (votação pública) interagindo com o serviço apenas por intermédio do *ContactServiceObject*. Em tais serviços, a criação baseia-se apenas na criação do próprio *CSCWServicesObject*. Já os serviços de construção completa, como os Serviços de Conferência Computadorizada, a construção abrange a criação dos objetos gerenciadores e representantes de membros e recursos.

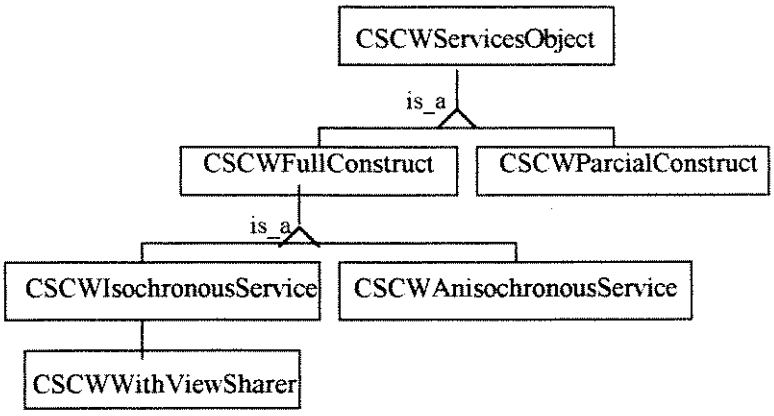


Figura 6.6 - Tipos especializados do *CSCWServicesObject*

Segundo a ação temporal entre os participantes de um grupo, um serviço cooperativo completo abrange duas classificações distintas assumindo o caráter *isócrono* ou o caráter *anisócrono*. O modo isócrono tem como preocupação principal o controle da concorrência dos canais de comunicação e dos recursos alocados. O caráter *anisócrono* se caracteriza com a não interação dos membros em tempo real, onde o gerenciamento visa estabelecer históricos e controlar somente o acesso aos recursos e canais intercambiáveis.

Finalmente, um serviço isócrono abre a possibilidade de um controle em tempo real de execução da qualidade assumida durante a sua evolução. Tal controle fica a cargo do objeto *ViewSharerObject* que, por sua vez, deverá ser conectado aos objetos gerenciadores de membros e recursos, além do próprio *CSCWServicesObject*, sendo neste item representado pelo objeto *CSCWWithViewSharer*.

Essa distinção funcional de *CSCWServices* garante o emprego da camada em todos os tipos de serviços cooperativos, onde foram cobertas as principais características, separando-as em objetos específicos derivados do *CSCWServicesObject*.

6.4.4 - Objeto ViewSharerObject

O objeto *ViewSharerObject* tem como objetivo acompanhar a evolução da qualidade de serviço do grupo cooperativo. Possui a autoridade de encerrar, interromper o grupo ou comunicar aos membros caso a qualidade atinja níveis inaceitáveis para a operacionalidade do grupo.

Informações a respeito da qualidade vigente no grupo cooperativo são fornecidas pelos objetos *CSCWServices*, *MembersManager* e *ResourcesManager*.

6.4.5 - Objeto MembersManagerObject

Representa o elo de comunicação entre todos os membros integrados por um certo serviço cooperativo. Visa gerenciar os membros de forma a conseguir uma seqüenciação na utilização dos canais de comunicação, além de proporcionar um interfaceamento entre o *CSCWServicesObject* e os membros. Quando a aplicação necessita de uma contínua verificação da qualidade de serviço, o *MembersManager* informa continuamente ao *ViewSharerObject* o estado dos membros sob o seu gerenciamento.

O objeto *MembersManagerObject* é responsável por gerenciar a utilização do canal formal de comunicação. A substituição do membro detentor do direito de utilização deste canal segue uma política pré-estabelecida na criação do serviço ou alterada no decorrer da ação cooperativa. Um membro é substituído quanto à utilização do canal formal de comunicação nas seguintes ocasiões:

- por tempo máximo de utilização
- pedidos de utilização de outros membros (inscrição para direito de voz)
- aleatoriamente, decidida pelo *chairman*

Caso ocorram vários pedidos para a utilização do canal formal, o *MembersManagerObject* pode valer-se de um dos seguintes critérios para o tratamento de conflito:

- por ordem de inscrição
- por ordem hierárquica
- pela intermediação do *chairman*

Como já descrito, um serviço cooperativo poderá assumir uma identidade isócrona ou anisócrona. O objeto gerenciador de membros (*MembersManagerObject*) também deverá suportar tais características por intermédio das especializações dos objetos *MembersManagerAnisochronous*, *MembersManagerIsochronous* e *MembersManagerWithViewSharer* (figura 6.7).

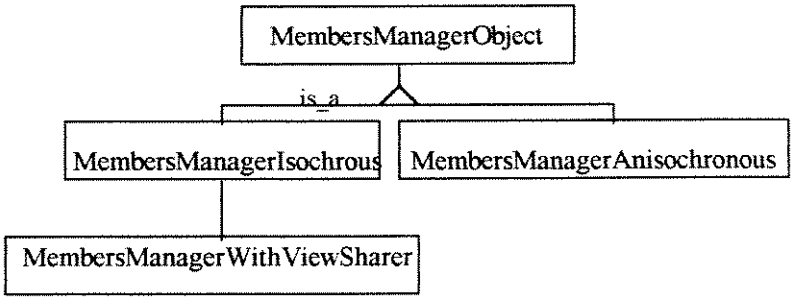


Figura 6.7 - Especializações do objeto *MembersManagerObject*

6.4.6 - Objeto MemberParticipantObject

A interação entre o mundo real e a camada cooperativa compete, a nível de cada usuário, ao objeto *MemberParticipantObject*. Para cumprir tal objetivo, sua instanciação consiste na replicação das informações externas à camada, tais como nome real, identificação do host e demais referências que o identifique.

Dessa forma, pode-se afirmar que para cada usuário integrado ao grupo cooperativo há exatamente uma instanciação do *MembersParticipantObject*. Tratando-se de um objeto cujo controle dá-se diretamente por cada usuário, permite-se o controle da qualidade local do serviço, onde as decisões tomadas às suas dependências (recursos locais) independem da autorização do grupo ou do *chairman*.

Cada membro é conectado ao serviço cooperativo no qual está integrado, por intermédio do *MembersManager*, que mantém um relacionamento entre eles. Objetos do tipo *MemberParticipantObject* manipulam os recursos, comuns ao grupo ou locais, através de mensagens enviadas ao *ResourcesManager*, podendo aproveitar os serviços oferecidos por servidores distintos.

As características cooperativas exigem uma real estrutura hierárquica de grupo, ou seja, deve-se alocar um membro que assumirá a responsabilidade pelo grupo, tomando, quando necessário, decisões pertinentes à manutenção do serviço. Uma política adotada é outorgar tal responsabilidade àquele que configurou o grupo, atribuindo-lhe o papel de *chairman* (*ChairmanMember*). Os demais participantes serão considerados unicamente como membros *normais* (*NormalMember*) restringindo-lhes os serviços que possam modificar a configuração ou o estado do serviço localmente.

As especializações do objeto do tipo *MemberParticipantObject* são representadas na figura 6.8.

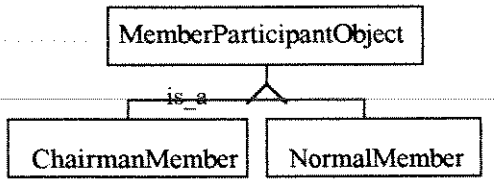


Figura 6.8 - Especializações do *MembersParticpantObject*

6.4.7 - Objeto ResourcesManagerObject

O controle de concorrência, não somente dos canais de comunicação, mas também dos recursos alocados para um determinado serviço cooperativo é de suma importância para uma fiel manutenção do mesmo. Desse modo, o objeto *ResourcesManagerObject* terá dois objetivos distintos:

- interfacear membros/recursos, representando um elo de comunicação entre os membros participantes e os recursos alocados para o serviço cooperativo. Essa tarefa é realizada tanto nos serviços que apresentam alocação dos recursos local quanto aqueles com recursos comum ao grupo (figura 6.9);

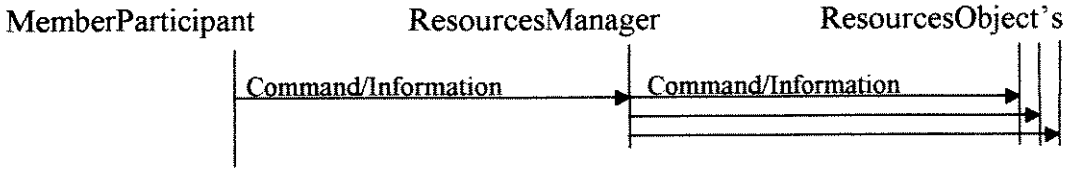


Figura 6.9 - Serviço Cooperativo com recursos locais

- controle de concorrência: esse tipo de controle é atuante apenas quando o serviço utiliza recursos comuns ao grupo, respeitando uma política estabelecida no momento da criação do serviço (figura 6.10).

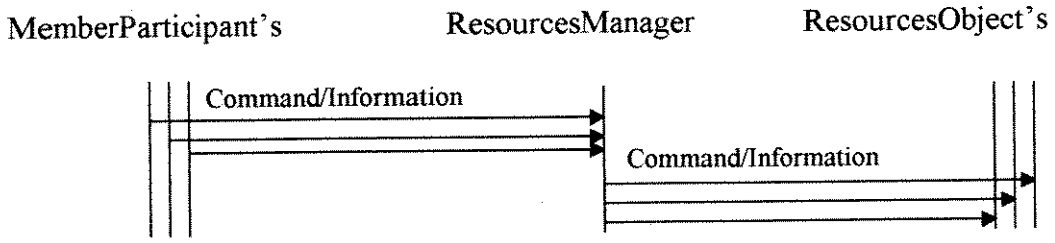


Figura 6.10 - Serviço Cooperativo com recursos comuns ao grupo

O *ResourcesManagerObject* visa também fornecer informações ao *ViewSharerObject* quanto às condições de uso dos recursos associados, permitindo dessa forma que se mantenha uma qualidade de serviço razoável para a sua manutenção.

Uma ação cooperativa envolve membros que poderão ou não estar dispersos geograficamente. Para caracterizar os dois cenários foram criadas, a partir do objeto *ResourcesManagerObject*, duas especializações, representadas pelas instanciações dos objetos *ResourcesManagerSharedResources* e *ResourcesManagerLocalResources*, respectivamente.

Os dois objetos especializados citados acima cobrem a maioria das características dos serviços a eles associados, porém, em virtude de fatores altamente restritivos de comportamento, são ainda especializados nos objetos *ResourcesManagerCoEdit* e *ResourcesManagerEMail*. A estruturação da hierarquia dos objetos derivados do *ResourcesManagerObject* é ilustrada na figura 6.11.

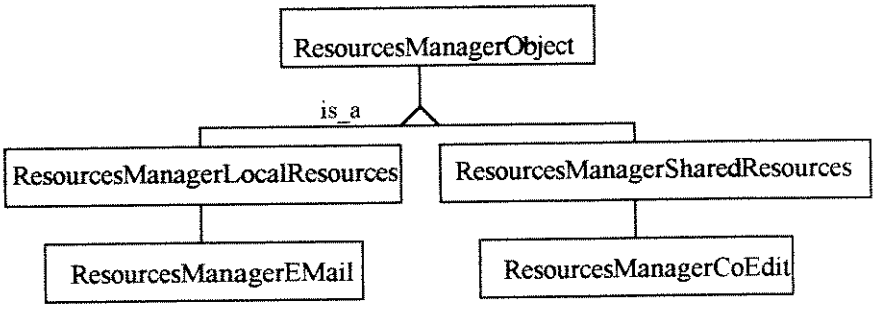


Figura 6.11 - Especializações do objeto *ResourcesManagerObject*

6.4.8 - Objeto ResourcesObject

Assim como os objetos que representam os membros participantes, os objetos do tipo *ResourcesObject* são referências diretas aos recursos, herdando todas as características a eles cabíveis e necessárias. Assim sendo, tais objetos estão intimamente relacionados com *portas*, *devices*, *gerenciadores de banco de dados*, *gerenciadores de janelas*, etc.

Para realizar o tratamento dos recursos, foram criados objetos do tipo *ResourcesObject*, gerenciados pelo *ResourcesManagerObject*, e especializados segundo cada tipo de recurso, comportando-se apenas como interfaces para os recursos específicos (figura 6.12).

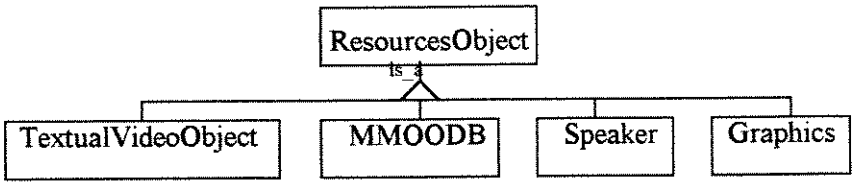


Figura 6.12 - Objetos representativos dos recursos

Na figura acima, são especificados os seguintes tipos de recursos:

- TextualVideoObject: vídeos que apenas manipulam informações na forma textual.
- MMOODBM (*Mulimidia Object Oriented Data-Base Manager*): Gerenciadores de Bases de Dados Orientadas a Objetos que manipulam informações multimídia
- Speaker: placas de som
- Graphics: vídeos que manipulam informações tanto na forma textual quanto na forma gráfica

6.5 - Exemplo de Utilização

Para melhor ilustrar a distribuição dos objetos, criou-se um exemplo de uso com quatro máquinas distintas, representando assim quatro pontos distintos geograficamente do serviço cooperativo. Conforme o exemplo, a máquina denominada de "*Host A*" desempenha a função de conter uma referência do objeto computacional básico centralizado *ContactServiceObject*, para o qual serão enviadas quaisquer mensagens e requisições a partir dos membros externos ao serviço (figura 6.13).

Ainda na figura 6.13, são apresentadas outras três máquinas ("*Host B*", "*Host C*" e "*Host D*") onde estão localizados os membros participantes do grupo cooperativo. A diferença entre estas três consiste no fato de que o "*Host B*" representa o *ChairmanMember*, portanto, engloba os objetos básicos de gerenciamento do serviço, constituídos pelo *CSCWServices-Server*, *MembersManagerObject* e *ViewSharerObject*. O "*Host B*" integra também as referências dos objetos gerenciador e interfaceador dos recursos locais alocados (*ResourcesManager-Object* e *ResourcesObject*). As referências dos objetos gerenciador e interfaceador dos recursos locais alocados são igualmente encontradas nos "*Host C*" e "*Host D*".

Caso o serviço se caracterizasse por recursos globais, o "*Host B*", o qual engloba o *chairman*, incluirá também em seu domínio o *ResourcesManager* e os objetos do tipo *ResourcesObject* para que sejam inteiramente comuns ao grupo. Neste último caso, os *hosts* que envolvem os membros que possuem como papel o caráter *NormalMember*, somente conterão o objeto *MemberParticipantObject*.

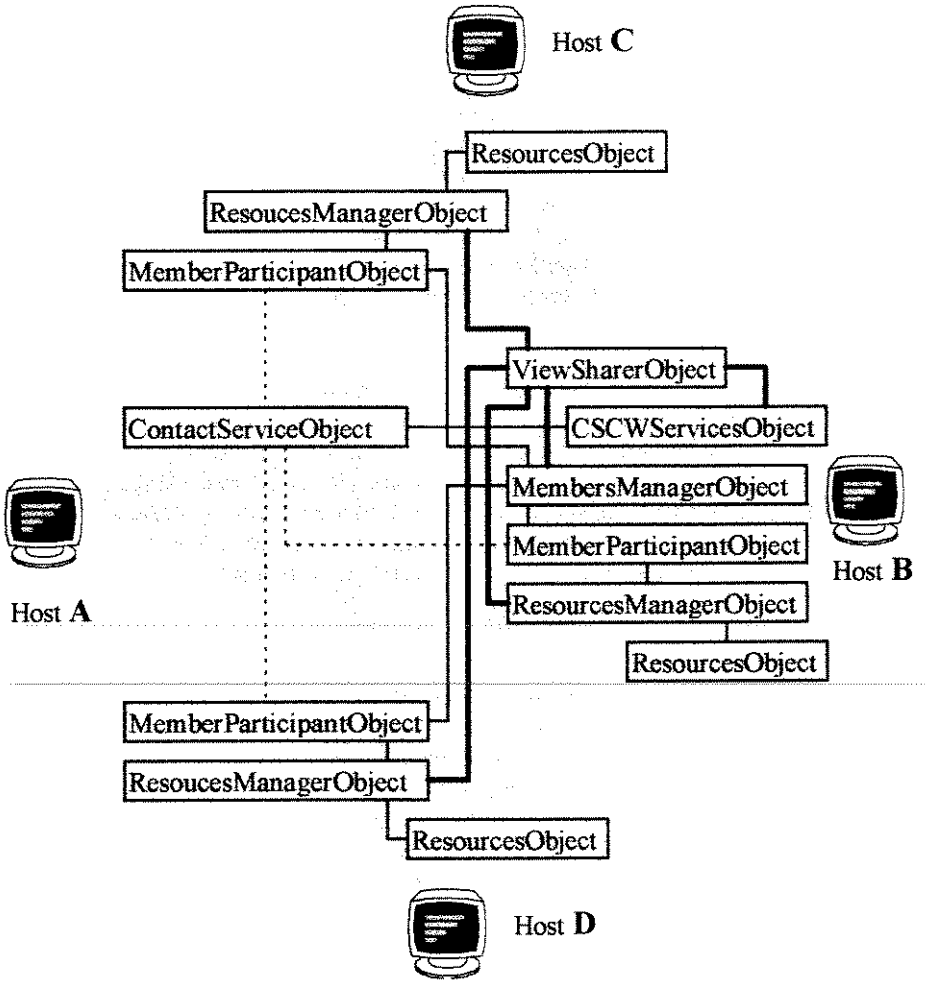


Figura 6.13 - Exemplo da distribuição dos objetos em um grupo com recursos locais

A linha pontilhada representa os contatos iniciais, tanto para o estabelecimento do grupo, quanto para o início das operações locais ou quaisquer contatos feitos quando os membros inclusos não passavam de membros externos ao grupo cooperativo.

O exemplo apresentado na figura 6.13 representa o modelo de aplicação cooperativa utilizado para o teste da Camada Groupware, onde a listagem de seu código fonte é mostrada no anexo "C".

Capítulo 7 - Aspectos de Implementação da Camada Groupware

7.1 - Introdução

Neste capítulo são abordados os detalhes referentes à implementação da Camada Groupware, tendo como principal objetivo a descrição dos atributos e métodos internos a cada objeto da camada.

Os métodos e os atributos neste capítulo explanados são provenientes do estudo das funcionalidades descritas no capítulo 6.

7.2 - Atributos dos Objetos da Camada Groupware

Para a descrição dos atributos e métodos presentes nos objetos da Camada Groupware, faz-se necessária a apresentação das constantes e estruturas definidas globalmente para a utilização nos objetos e aplicações associadas à Camada Groupware.

Neste tópico serão relatadas, nos ítens 7.2.1 a 7.2.3, as constantes e estruturas utilizadas na implementação e exportadas pela Camada Groupware e, nos ítens 7.2.4 a 7.2.10 os atributos presentes nos objetos pertencentes à referida camada.

No caso das especializações de objetos, são apresentados apenas os atributos que distinguem um objeto-filho de seu antecessor na árvore hierárquica.

7.2.1 - Constantes Presentes na Camada Groupware

- *MAX_GROUPS* → quantidade máxima de grupos abertos
- *MAX_MEMBERS* → número máximo de membros distribuídos em um grupo
- *MAX_POLICIES* → tamanho limite do vetor de políticas
  MAX_RESOURCES → número máximo de recursos alocados (locais ou globais)
- *MAX_INF_CHANNELS* → quantidade limite para canais informais abertos

7.2.2 - Tipos Enumerados da Camada Groupware

- *service_status* → CREATING,CREATED,RUNNING
- *members_status* → NOACTIVE,ACTIVE,SLEEP
- *cscw_types* → PARCIAL, ISOSERV, ANISERV, WVSSERV
- *memman_types* → ISO,ANI,WVS
- *mempar_types* → CHAIRMAN,NORMAL
- *resman_types* → LOCAL,SHARED,COEDIT,EMAIL
- *resour_types* → TEXTUAL_VIDEO

- *channel_types* → FORMAL,GENMESS,INFORMAL

7.2.3 - Estruturas Presentes na Camada Groupware

- *T_Policy* → caracteriza uma política específica
  - *PolicyName* → nome da política
  - *Value* → valor para a referida política

- *T_GeneralObject* → identificação de um objeto CORBA
  - *ObjectName* → nome do objeto
  - *ObjectHost* → host do objeto
  - *ObjectType* → tipo do objeto (especializações)

- *T_Time* → estrutura para horário
  - *Hour* → hora
  - *Minutes* → minutos

- *T_DataTime* → estrutura para data/hora
  - *Day* → dia
  - *Month* → mês
  - *Year* → ano
  - *DayTime* → estrutura para hora/minutos

quando os ítens são setados em 0 (zero) o chairman assume todas as operações envolvendo ações temporais

- *T_Member* → referência a um membro do grupo
  - *MemberName* → nome do membro
  - *MemberHost* → localização do membro
  - *MemberRole* → papel desempenhado pelo membro
  - *QoS* → qualidade de serviço
  - *MemberType* → tipo de membro para acesso da plataforma CORBA

- *T_Service* → descrição de um serviço
  - *ServiceName* → nome do serviço criado
  - *ServiceComment* → pequeno comentário a sobre o serviço
  - *Chairman* → chairman do grupo
  - *NumberOfPolicies* → número de políticas no vetor de políticas
  - *ServicePolicies* → vetor de politicas

- *T_Resources* → referencia um recurso alocado
  - *ResourceName* → nome do recurso
  - *ResourceHost* → localização
  - *ResourceType* → tipo do recurso
  - *Formal* → identifica um canal de comunicação

(tipo)

MinQos → qualidade estabelecida para o recurso

- *T_ResourcesManager* → atributos de um *ResourcesManager*
  Resources → relação de recursos alocados neste gerenciador
  NumberOfResources → número de recursos na lista acima
  MinQoS → qualidade mínima do gerenciador
  ResManType → especifica o seu tipo

7.2.4 - Atributos do Objeto ContactServiceObject

- *Proper* → apenas a descrição do próprio objeto.
- *FactoringList* → lista de grupos em construção
- *CreatedList* → relação de grupos criados
- *Chairman* → identificação do chairman
- *CSCW* → ponteiro para o *CSCWServicesObject*
- *NumberOfCreated* → número de grupos criados
- *NumberOfFactoring* → quantidade de grupos em construção

7.2.5 - Atributos do Objeto ServiceFactoryObject

- *Proper* → identificação do próprio objeto
- *ContactID* → descrição do *ContactServiceObject*
- *CSCWServicesID* → identificação do *CSCWServicesObject*

7.2.6 - Atributos do Objeto CSCWServicesObject

- *CSCWServicesObject*
  - *Proper* → definição do próprio objeto
  - *ContactID* → identificação do *ContactServiceObject*
  - *ChairmanID* → informações a respeito do *chairman*
  - *Contact* → ponteiro para o *ContactServiceObject*
  - *ServiceName* → nome do serviço em questão
  - *ServiceComment* → comentário a respeito do serviço
  - *ServiceStatus* → estado corrente do serviço

- *CSCWParcialConstruct*
  nenhum atributo que o diferencie da classe-mãe

- *CSCWFullConstruct*
  - *MembersManagerID* → identificação do objeto *MembersManagerObject*
  - *MemMan* → ponteiro para o objeto *MemberManagerObject*

- *CSCWIsochronousService*
  - *StartTime* → momento para o início da ação cooperativa
  - *DurationTime* → tempo de duração do serviço

- *CSCWAnisochronousService*
  nenhum atributo que o diferencie da classe-mãe

- *CSCWServiceWithViewSharer*
  - *ViewSharerID* → identificação do *ViewSharerObject*
  - *View* → ponteiro para o *ViewSharerObject*

7.2.7 - Atributos do Objeto MembersManagerObject

- *MembersManager*
  - *Proper* → definição do próprio objeto
  - *CSCWServicesID* → descrição do objeto *CSCWServices*
  - *MembersList* → lista de membros conectados ao *MembersManager*
  - *Questions* → questões pendentes (serviço de votação)
  - *MemPar* → lista de ponteiros para os *MemberParticipant's*
  - *CSCW* → ponteiro para o *CSCWServices*
  - *NumberOfMembers* → número de membros conectados
  - *NumberOfPolicies* → número de políticas associadas ao objeto
  - *NumberOfQuestions* → quantidade de questões pendentes

- *MembersManagerIsochronous*
  - *FormalChannel* → identificação do canal formal de comunicação
  - *CurrentMember* → membro detentor da utilização do canal formal
  - *MaxUtilizationTime* → tempo máximo de utilização contínua do canal formal

- *MembersManagerAnisochronous*
  nenhum atributo que o diferencie da classe-mãe

- *MembersManagerWithViewSharer*
  - *ViewSharerID* → descrição do objeto *ViewSharer* anexado ao grupo

UNICAMP
BIBLIOTECA CENTRAL

7.2.8 - Atributos do Objeto MemberParticipant

- *MemberParticipant*
  - *MemberID* → Identificação do membro sob representação
  - *Proper* → Estrutura de identificação do objeto
  - *ResourcesManagerID* → Descrição do *ResourcesManager* associado
  - *MembersManagerID* → Informações sobre o *MembersManager*
  - *FormalChannelID* → Descrição do canal formal de comunicação

    InformalChannelsID → Lista de identificação dos canais informais abertos
  - *MemMan* → Ponteiro para o *MembersManager*
  - *ResMan* → Referência para o *ResourcesManager*
  - *InfoChannels* → Lista de ponteiros para os canais informais
  - *ResourcesManagerType* → Tipo do *ResourcesManager*
  - *MembersManagerType* → Informação sobre o tipo do *MembersManager*
  - *NumberOfInformalChannels* → Número de canais informais abertos
  - *MinQoS* → Qualidade mínima estabelecida para o membro
  - *Mstatus* → Estado do membro (*ACTIVE, NOACTIVE, etc)*

- *ChairmanMember*

  nenhum atributo que o diferencie da classe-mãe

- *NormalMember*

  nenhum atributo que o diferencie da classe-mãe

7.2.9 - Atributos do Objeto ResourcesManagerObject

- *ResourcesManager*
  - *ResourcesManagerID* → Descrição específica do objeto *ResourcesManager*
  - *Proper* → Identificação geral do objeto
  - *Resources* → Lista de ponteiros para os recursos alocados

- *ResourcesManagerLocalResources*
  - *MemberProprietary* → Identificação do membro proprietário
  - *MemPar* → Ponteiro para o membro proprietário

- *ResourcesManagerSharedResources*
  - *MembersProprietaries* → Lista de membros proprietários

- *MemPar* → Ponteiros para os membros proprietários
- *NumberOfMembers* → Número de membros utilizando o gerenciador de recursos

- *ResourcesManagerEMail*
  nenhum atributo que o diferencie da classe-mãe

- *ResourcesManagerCoEdit*
  nenhum atributo que o diferencie da classe-mãe

7.2.10 - Atributos do Objeto ResourcesObject

- *ResourcesObject*
  - *MinQoS* → qualidade de serviço mínima para o recurso
  - *Proper* → descrição do objeto
  - *ResourcesManagerID* → descrição do objeto *ResourcesManager*

- *TextualVideo*
  - *TVDisplay* → estrutura *Display* para controlar a comunicação da janela com a estação
  - *TVWindow* → identifica a janela (estrutura *Window*)
  - *TVGrContext* → definição do *graphic context* (estrutura *GC*)
  - *TVHint* → tamanho e posição da janela (estrutura *XSizeHints*)
  - *TVWmHint* → estrutura para definição do estado inicial, ícones, grupos, etc (estrutura *XWMHints*)
  - *TVEvent* → eventos passíveis de acontecimento (estrutura *XEvent*)
  - *TVForeground* → cor de frente
  - *TVBackground* → cor de fundo
  - *TVScreen* → número do *screen* associado

7.3 - Métodos Implementados

Em todos os métodos relacionados abaixo, o último parâmetro consiste em uma referência para a estrutura do tipo CORBA::Environment do *Orbix* (IT_env) destinada à checagem de falhas de comunicação entre os objetos distribuídos pelo ambiente computacional.

7.3.1 - ContactServiceObject

7.3.1.1 - *ConstructNewService*

A criação de um serviço pode ser resumido na criação dos objetos básicos de gerenciamento e de representação dos elementos físicos (membros e recursos) e no processo de conexão entre membros e recursos, desde que sejam obedecidas certas políticas de organização e de manutenção. Esta fase pode compreender, superficialmente falando, duas etapas: a primeira, não obrigatória, consiste no convite aos possíveis membros da comunidade que poderão participar do serviço em questão e, a segunda, baseia-se na criação propriamente dita do serviço.

Estas duas possíveis etapas são tratadas no objeto *ServiceFactoryObject* que irá encarregar-se da seqüenciação da criação de um serviço cooperativo, para isso o método *ConstructNewService* tem como função a construção do referido objeto para que este possa assumir o controle da criação do serviço (figura 7.1).

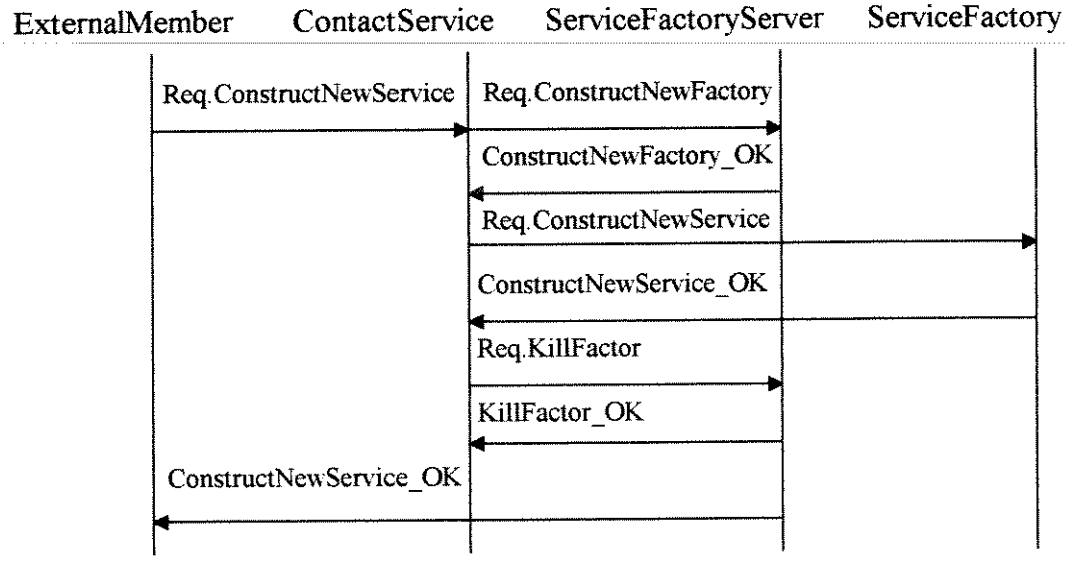


Figura 7.1 - Criação de um objeto *ServiceFactoryObject*

Protótipo:

ConstructNewGroup(ServiceType,ServiceName,Comment,StartTime,
DurationTime,TypeOfMembersManager,MembersList,
MembersNumber, ResourcesManagerID,PoliciesVector,
PoliciesNumber,IT_X)

Parâmetros:

- ServiceType (short): Tipo do serviço cooperativo a ser confeccionado (PARCIAL, ISOSERV, ANISERV, WVSSERV). Parâmetro de entrada.
- ServiceName (const char *): Nome do grupo a ser aberto. Parâmetro de entrada.

- Comment: (const char *) Breve descrição do grupo. Argumento de entrada.
- StartTime (T_DataTime): Hora para o início. Parâmetro de entrada.
- DurationTime: (T_Time) Duração do cooperativismo. Parâmetro de entrada.
- TypeOfMembersManager (short): Tipo de gerenciamento estabelecido para os membros (ISO,ANI,WVS). Parâmetro de entrada.
- MembersList (_IDL_SEQUENCE_T_Member): Relação dos membros que irão compor, inicialmente, o grupo cooperativo. Parâmetro de entrada.
- MembersNumber: (short) Número de membros inscritos na lista. Parâmetro de entrada.
- ResourcesManagerID (_IDL_SEQUENCE_T_ResourcesManager): Define o núcleo de gerenciamento de recursos, incluindo a relação dos próprios recursos que irão ser manipulados. Parâmetro de entrada.
- PoliciesVector: (_IDL_SEQUENCE_T_Policy) Vetor de políticas adotadas para o serviço. Parâmetro de entrada.
- PoliciesNumber (short): Número de políticas presentes no vetor. Parâmetro de entrada.
- Retorno da função (short): Indicativo de erro durante a fase de criação.
- *IT_env* (IT_X): estrutura *Orbix* para tratamento de erros

7.3.1.2 - *CloseGroup*

O fechamento de um serviço, quando tratado neste objeto (*ContactServiceObject*), implica que toda a transação e negociação para tal será feita no a partir do objeto do tipo *CSCWServicesObject*, representando o grupo que a ser encerrado.

Sendo assim, esse método irá simplesmente requisitar ao *CSCWServicesObject* o seu auto fechamento, sendo que, após ter sido consumado o processo, o *ContactServiceObject* evocará o método *DeleteCSCWServices*, presente no objeto *CSCWServicesServer* conforme é ilustrado na figura 7.2.

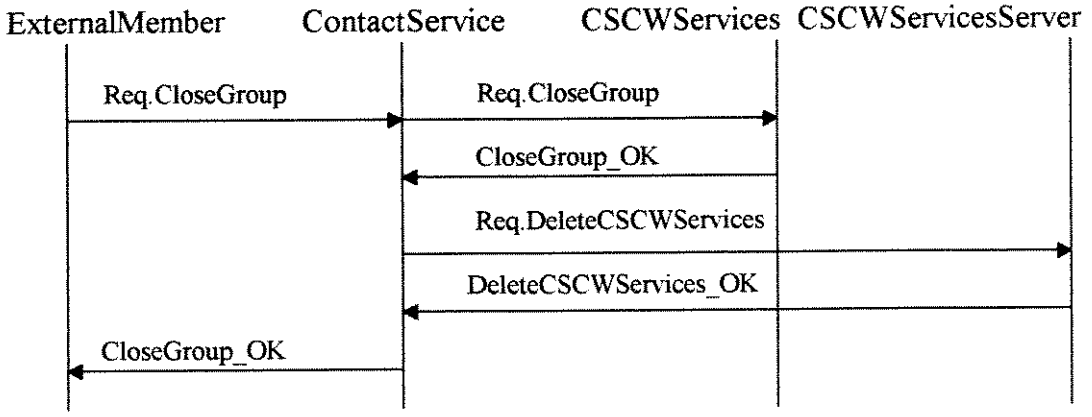


Figura 7.2 - Fechamento de um serviço cooperativo

Protótipo:

CloseGroup(GroupName,IT_X)

Parâmetros:

- GroupName (const char *): Nome do grupo a ser fechado. Parâmetro de entrada.
- *IT_env (IT_X)*: Estrutura *Orbix* para tratamento de erros
- Retorno da função (short): Indicativo de erro de fechamento.

7.3.1.3 - *SearchGroup*

O presente método visa facilitar o processo de intercâmbio entre os membros externos e os grupos cooperativos já criados, devolvendo o nome do *chairman* e sua localização, além do número identificador do grupo, que em termos de implementação corresponde à posição dentro da lista de grupos criados (*CreatedList*). A figura 7.3 mostra a execução desse método.

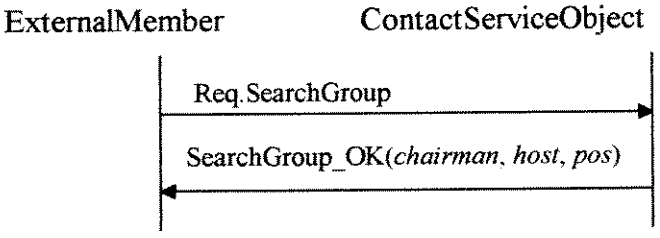


Figura 7.3 - Obtenção de informações básicas de um grupo criado

Protótipo:

SearchGroup(GroupName,ChairmanName,GroupHost,GroupPositionVector, IT_X)

Parâmetros:

- GroupName (const char *): Nome do grupo a ser procurado. Parâmetro de entrada.
- ChairmanName (char *): Nome do chairman do referido grupo. Parâmetro de saída
- GroupHost (char *): Localização do grupo (host). Parâmetro de saída.
- GroupPositionVector (short): Posição referenciada na lista de grupos criados, interna ao objeto *ContactService*. Parâmetro de saída.
- Retorno da função (short): Indicativo de erro.
- IT_env (IT_X): estrutura *Orbix* para detecção de erros

7.3.1.4 - *ActiveGroup*

Após a etapa de criação de um serviço, este ainda não se encontra propriamente em execução, pois o estabelecimento, com o respectivo acordo entre as partes integrantes do serviço, ocorre previamente à própria evolução do cooperativismo.

Portanto, a iniciação de um serviço obedece uma das políticas (figura 7.4):

- início automático: sendo o momento definido no momento da contactação do *chairman* com os possíveis membros participantes. Neste caso, não ocorre a atuação do *ContactServiceObject*.

- aleatório: onde o *chairman* poderá iniciar o serviço a qualquer momento, desde que a etapa de estabelecimento tenha sido completada com êxito. A requisição é realizada por intermédio do *ContactServiceObject*.

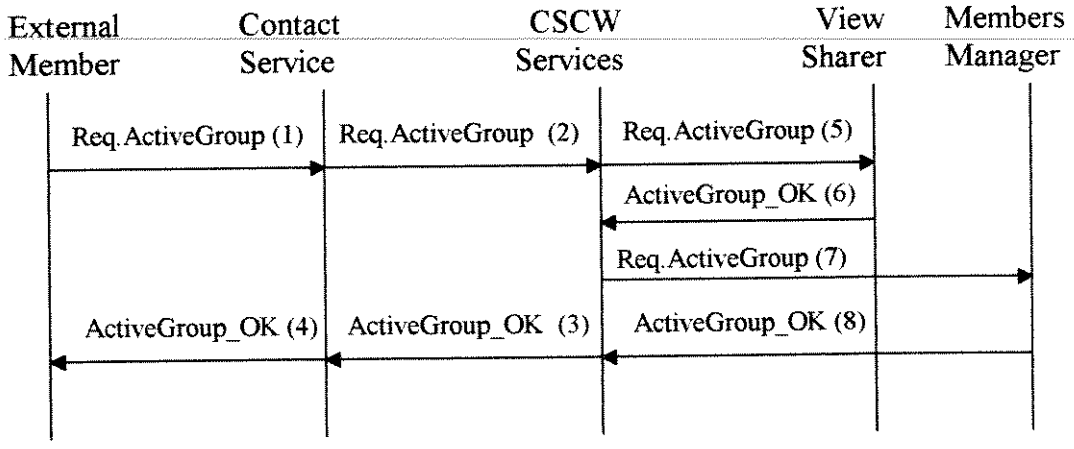


Figura 7.4 - Início da Ação Cooperativa

A numeração da figura acima informa que:

1. mensagens (1), (2), (3) e (4): ausentes quando o início é automático a partir do *CSCWServicesObject*
2. mensagens (5) e (6): ausentes quando o serviço não exige a presença do *ViewSharerObject*
3. mensagens (5), (6), (7) e (8): ausentes quando o serviço é de construção parcial

Protótipo:

ActiveGroup(GroupPositionVector,IT_X)

Parâmetros:

- GroupPositionVector (short): Indica o posicionamento que o serviço em questão se encontra na lista de grupos criados, interna ao objeto

ContactServiceObject. Essa posição é obtida pelo método auxiliar *SearchGroup*. Parâmetro de entrada.
- *IT_env (IT_X):* estrutura *Orbix* para checagem de erros
- Retorno da função (short): Indica ocorrência de erro na ativação.

7.3.1.5 - *InitMember*

Para um melhor gerenciamento e liberdade de ação, os membros terão que, após o grupo a qual pertencerão seja completamente confeccionado e iniciado (ver *ActiveGroup*), se "logarem" no mesmo. Essa etapa de iniciação individual de cada membro é realizada através do método *InitMember* conforme mostra a figura 7.5.

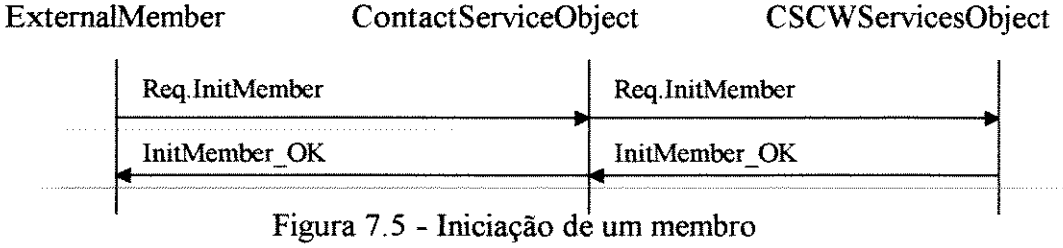


Figura 7.5 - Iniciação de um membro

Protótipo:

InitMember(GroupPositionVector,IT_X)

Parâmetros:

- GroupPositionVector (short): Indica o posicionamento que o serviço em questão se encontra na lista de grupos criados, interna ao objeto *ContactServiceObject*. Essa posição é obtida pelo método auxiliar *SearchGroup*. Parâmetro de entrada.
- *IT_env (IT_X)*: estrutura *Orbix* para tratamento de falhas
- Retorno da função (short): Indica ocorrência de erro na ativação.

7.3.1.6 - *AddNewMember*

Um membro externo à um serviço poderá pedir a sua anexação ao grupo mediante o envio de uma mensagem ao *ContactServiceObject* que a tratará de acordo com a política de entrada de novos membros vigente.

A anexação de um novo membro poderá se dar de uma das três formas:

- livremente, sem vínculo a qualquer autorização;
- por um sistema de votação de todos os membros participantes do serviço;
- pela permissão unicamente do *chairman*.

O processo de entrada de um novo membro será mais detalhadamente explicado no momento da descrição do método *AddNewMember*, pertencente ao objeto *CSCWServicesObject*. No momento, o contexto de anexação se dará apenas na comunicação entre o *ContactServiceObject* e o *CSCWServiceObject*, como mostrado na figura 7.6.

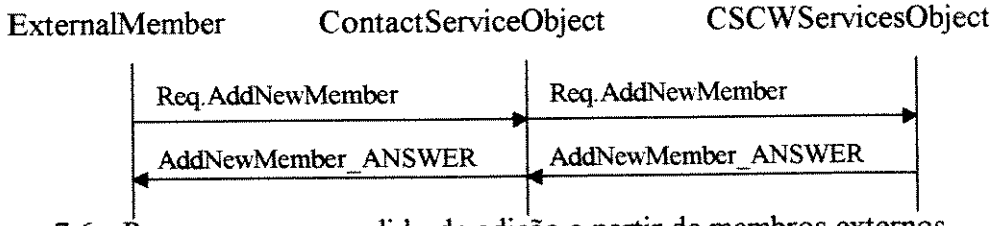


Figura 7.6 - Processo para o pedido de adição a partir de membros externos

Após o pedido de anexação ao grupo, o membro deverá, caso aceito, proceder a etapa de iniciação quando no momento de sua efetiva anexação ao grupo cooperativo (ver *InitMember*).

Protótipo:

AddNewMember(MemberIdentification,GroupName,ResourcesManagerID, ResourcesNumber,IT_X)

Parâmetros:

- MemberIdentification (T_Member): Estrutura que contém as informações do membro a ser adicionado. Parâmetro de entrada.
- GroupName (const char *): Nome do grupo no qual o referente membro irá ser anexado. Parâmetro de entrada.
- ResourcesManagerID (T_ResourcesManager): Relaciona os recursos que integrarão o ambiente do novo membro. Parâmetro de entrada.
- ResourcesNumber: Número de recursos constantes na lista. Parâmetro de entrada.
- *IT_env (IT_X)*: Estrutura *Orbix* para tratamento de falhas
- Retorno da função (short): Erro na anexação.

7.3.1.7 - *GetCreatedInfo*

Uma importante consulta que poderá ser realizada pelos membros da comunidade (fazendo ou não parte de um dado serviço), consiste na obtenção de uma relação dos serviços abertos em um determinado momento.

As mensagens trocadas entre o membro e o objeto consistem apenas na requisição do método e, como retorno, a descrição do serviço, contendo o nome do grupo, a sua localização e a identificação de seu *chairman*. A figura 7.7 ilustra tal método.

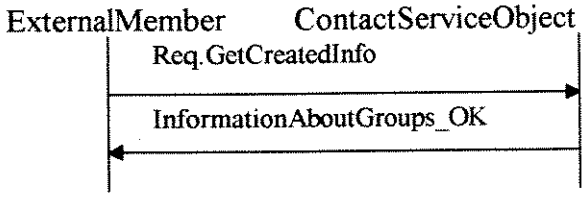


Figura 7.7 - Obtenção de informações sobre grupos criados

Protótipo:

GetCreatedInfo(GroupInformations,CreatedGroupNumber,IT_X)

Parâmetros:

- GroupInformations (_IDL_SEQUENCE_T_Member): Lista contendo as informações dos grupos cooperativos já criados. Tais informações consistem no nome do grupo, host sobre o qual se encontra o objeto *CSCWServices* do referido grupo e o nome de seu *chairman*. Parâmetro de saída.
- CreatedGroupNumber (short): Número de grupos criados referenciados na lista. Parâmetro de saída.
- *IT_env (IT_X)*: Estrutura *Orbix* para tratamento de falhas

7.3.2 - ServiceFactoryObject

7.3.2.1 - *ConstructNewService*

Após ter ocorrida a certificação de que o serviço poderá ser criado mediante o preenchimento de todos os papéis criados e a garantia de se alcançar a QoS mínima para a sua manutembilidade, a etapa de criação do serviço alcança uma nova fase, constituida pelos seguintes passos:

- criação dos objetos *CSCWServicesObject* e demais objetos das sub-camadas inferiores
- ligação do *CSCWServicesObject* recém criado objeto ao *ContactServiceObject*
- auto-destruição do objeto *ServiceFactoryObject*

Realizados os passos acima, o serviço passará do estado *em confecção* para o *pronto para a execução*, esperando apenas pela requisição de seu início. O processo de criação de um objeto do tipo *CSCWServicesObject* é ilustrado na figura 7.8.

As mensagens numeradas de (1) e (2) não estarão presentes quando o grupo que irá ser aberto for do tipo *CSCWParcialConstruct*.

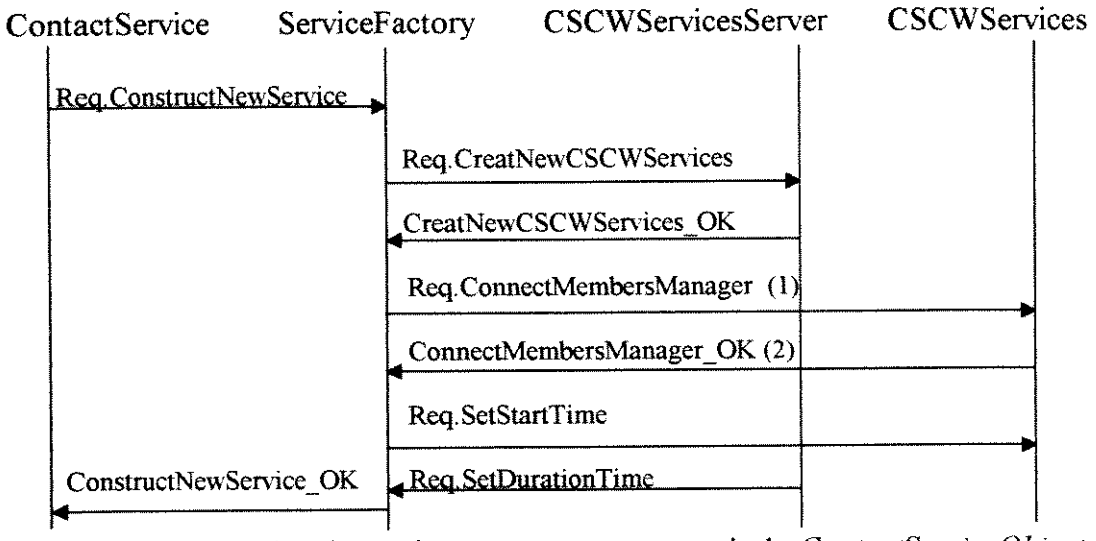


Figura 7.8 - Etapa de criação de um novo grupo a partir do *ContactServiceObject*

Protótipo:

ConstructNewService(cscwtype, servname, comment, start, duration, memmantype, members, nummembers, resmanID, policies, npol, IT_env)

Parâmetros

- *cscwtype* (short): especifica o tipo de serviço
- *servname* (const char *): nome do serviço a ser construído
- *comment* (const char *): breve comentário a respeito do serviço
- *start* (const T_DataTime&): data/horário para início do cooperativismo (setado em zero representa início manual pelo *chairman*)
- *duration* (const T_Time&): duração do grupo (setado em zero indica a finalização pelo *chairman* do serviço)
- *memmantype* (short): tipo do gerenciador de membros
- *members* (const _IDL_SEQUENCE_T_Member&): relação de membros que serão incorporados no momento da criação do grupo
- *nummembers* (short): número de membros a incorporar
- *resmanID* (const _IDL_SEQUENCE_T_ResourcesManager&): lista de descrição para os *ResourcesManagerObject*
- *policies* (const _IDL_SEQUENCE_T_Policy&): relação de políticas do serviço
- *npol* (short): quantidade de políticas contidas no vetor acima
- *IT_env*: estrutura *Orbix* para tratamento de erros

7.3.3 - CSCWServices

7.3.3.1 - *CSCWServices::SetStatus*

Um serviço cooperativo é caracterizado, dentre outras coisas, por seu estado corrente. Assim sendo, poderão ser identificados os estados quando durante o processo de configuração (*CREATING*), quando pronto para execução (*CREATED*) ou no estado setado quando o mesmo se encontra em execução (*RUNNING*).

O método *SetStatus* auxilia o processo administrativo do *CSCWServices* pelo fato de ser o responsável de setar tal atributo, sendo sua ativação por intermédio do *ContactService* (figura 7.9).

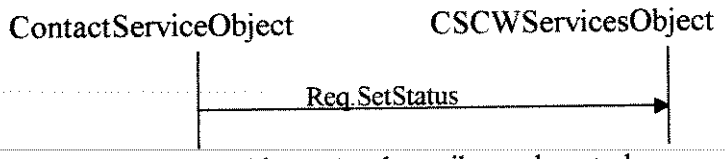


Figura 7.9 - Alteração do atributo de estado

Protótipo:

SetStatus(newstatus,IT_env)

Parâmetros

- *newstatus* (short): novo estado a ser estabelecido para o serviço
- *IT_env*: estrutura para consistência de falhas

7.3.3.2 - *CSCWServices::GetStatus*

Apenas retorna o estado atual do serviço cooperativo para qualquer consulta realizada pelo *ContactServiceObject*, conforme ilustra a figura 7.10.

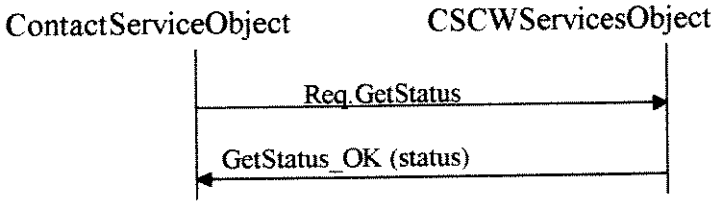


Figura 7.10 - Captação do atributo de estado

Protótipo:

GetStatus(IT_env)

Parâmetros

- *IT_env*: estrutura para detecção de falhas
- retorno da função (short): valor do estado corrente

7.3.3.3 - *CSCWParcialConstruct::ActiveGroup*

Após a etapa de configuração de um seviço cooperativo, o mesmo se encontra ainda no estado *CREATED*, aguardando que seu início seja realizado automaticamente, quando setado o atributo *StartTime*, ou aleatoriamente a pedido do chairman do grupo, quando o referido atributo se encontra resetado (valores zero em seus campos).

Por apresentar uma construção parcial, este tipo de serviço não realizará conexões com objetos gerenciadores, administradores ou agentes, portanto, apenas atualizará o seu atributo de estado.

Protótipo:

ActiveGroup(contact,IT_env)

Parâmetros

- *contact* (ContactService *): ponteiro para o *ContactServiceObject*
- *IT_env*: estrutura *Orbix* para checagem de erros

7.3.3.4 - *CSCWIsochronousService::ActiveIGroup/ CSCWAnisochronousService::ActiveAGroup / CSCWServiceWithViewSharer::ActiveVGroup*

Por sua vez, ao contrário dos serviço de construção parcial, aqueles de plena feitura, quando ativos os métodos de ativação, o estado do serviço passará a *RUNNING* por intermédio da abertura da conexão entre os objetos da camada, aproveitando os atributos de identificação daqueles limítrofes.

Os três métodos descritos realizam a mesma operação, porém em tipos de objetos (especiaizações) diferentes. A forma genérica de representação dos mesmos é encontrada na figura 7.11.

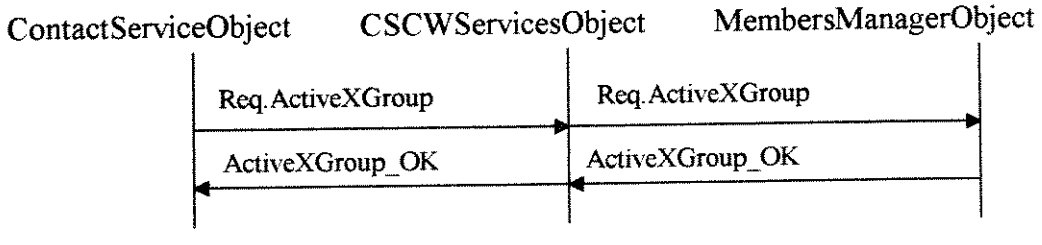


Figura 7.11 - Processo de ativação de um grupo

Protótipo:

ActiveIGroup(contact,IT_env)
ActiveAGroup(contact,IT_env)
ActiveVGroup(contact,IT_env)

Parâmetros:

- *contact* (ContactService *): ponteiro para o *ContactServiceObject*
- *IT_env*: estrutura *Orbix* para checagem de erros

7.3.3.5 - *CSCWFullConstruct::ConnectMembersManager*

Durante a configuração de um grupo cooperativo cuja construção é integral, ou seja, com a instanciação de todos os objetos gerenciadores, administradores e agentes, surge a necessidade de criação daqueles localizados nos níveis hierárquicos abaixo do *CSCWServicesObject*, representados pelos objetos *MembersManager*, *MemberParticipant*, *ResourcesManager* e *ResourcesObject*.

O método *ConnectMembersManager* terá a função de criar um objeto gerenciador de membros e, a partir deste, requisitar a construção e anexação dos demais objetos integrantes na camada *Groupware*.

A referida criação é feita através de chamadas seqüenciais, onde cada objeto é responsável por seu nível inferior. Desta forma, a construção será finalizada com sucesso se, e somente se, todas as instanciações das camadas inferiores forem também satisfatórias.

A figura 7.12 ilustra a etapa de anexação do *MembersManager* a partir de uma requisição do *ContactServiceObject*.

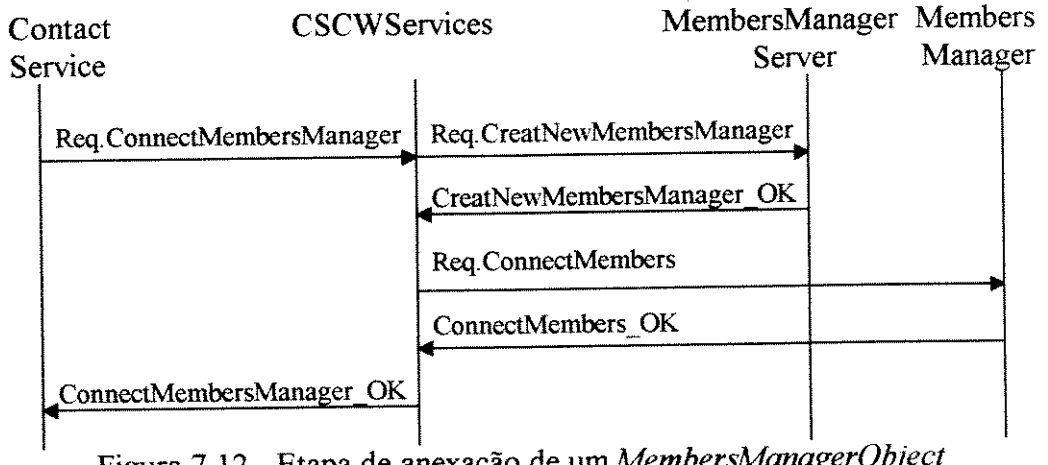


Figura 7.12 - Etapa de anexação de um *MembersManagerObject*

Protótipo:

ConnectMembersManager(servicename, memmantype, resmanID, members, nummembers, policies, npol, IT_env)

Parâmetros:

- *servname* (const char *): nome do serviço a ser construído
- *memmantype* (short): tipo do *MembersManagerObject* a ser anexado
- *resmanID* (const _IDL_SEQUENCE_T_ResourcesManager&): lista de descrição para os *ResourcesManagerObject*
- *members* (const _IDL_SEQUENCE_T_Member&): relação de membros que serão incorporados no momento da criação do grupo
- *nummembers* (short): número de membros a incorporar
- *policies* (const _IDL_SEQUENCE_T_Policy&): relação de políticas do serviço
- *npol* (short): quantidade de políticas contidas no vetor acima
- *IT_env*: estrutura *Orbix* para tratamento de erros

7.3.3.6 - *CSCWFullConstruct::InitMember*

Tratando-se de ações cooperativas, envolvendo pessoas em tempos e espaços distintos, torna-se desaconselhável que os membros participantes de um grupo tornem-se ativos assim que o mesmo for ativado pelo chairman ou automaticamente pelo próprio *CSCWServices*.

Para evitar esse fato que, talvez poderia provocar problemas na comunicação, com possíveis desentendimentos e desencontros, a camada possui um serviço de iniciação individual de cada membro.

A função de tal método é propagar a requisição, oriunda da interface externa do serviço e transmitida por intermédio do *ContactService*, até o *MembersManager* para que este possa tomar as devidas providências para instanciar os descritores representativos de tal membro, assim como conectá-lo aos recursos alocados (figura 7.13).

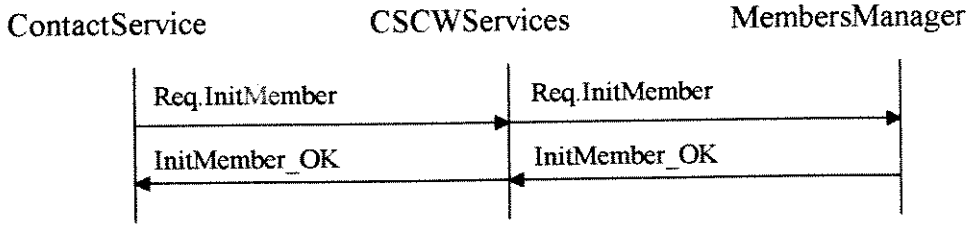


Figura 7.13 - Iniciação de um membro

Conforme é ilustrado na figura acima (7.13), o *CSCWServices* apenas interfacea a requisição do *ContactService* até o *MembersManager* para que possa ser garantida a modularidade especificada da camada.

Protótipo:

InitMember(member,IT_env)

Parâmetros:

- *member* (const char *): nome do membro a ser ativado
- *IT_env*: utilizada para verificação de falhas

7.3.3.7 - *CSCWFullConstruct::AddNewMember*

Assim como a iniciação de um membro, o método destinado à introdução de um membro externo ao grupo ficará a cargo do *MembersManager*. Aliás, qualquer operação sobre os objetos do tipo *MemberParticipant*, será de sua incumbência.

O tratamento é análogo ao exibido para o método *InitMember*, conforme ilustra a figura 7.14.

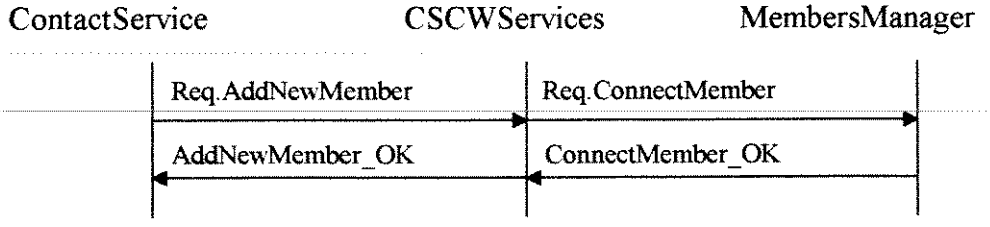


Figura 7.14 - Anexação de um novo membro

Maiores detalhes poderão ser encontrados quando na descrição do objeto *MembersManager*, referente ao método *ConnectMember*.

Protótipo:

AddNewMember(member, nserv, resmanID, IT_env)

Parâmetros:

- *member* (const T_Member&): identifica o membro a ser anexado
- *nserv* (const char*): nome do serviço corrente
- *resmanID* (const T_ResourcesManager&): descrição do *ResourcesManager* associado conjuntamente aos recursos alocados (caso recursos locais)
- *IT_env*: estrutura para verificação de falhas

7.3.3.8 - *CSCWFullConstruct::CloseGroup*

Sendo a idéia básica de um serviço cooperativo o envolvimento de várias pessoas, podendo compartilhar recursos e se interagirem em um mesmo instante ou em momentos distintos, existe uma política específica, assim como para a abertura, também para o encerramento de um serviço cooperativo. Em primeira intância, o encerramento poderá ocorrer:

- imediatamente após o pedido
- após um certo tempo

Independente do tempo, o requerimento poderá ser originado do *chairman*, quando este achar o momento oportuno, ou pelo *ViewSharerObject*, quando a QoS abaixar a níveis inaceitáveis. Dessa forma, poderemos ter:

- Fechamento requisitado pelo *chairman*:
  → discutida por votação entre os membros (definir anteriormente o porcentual de aprovação)
  → imposta (sem direito à votação)

- Requisitada pelo *ViewSharerObject*
  → passada a responsabilidade ao *chairman*
  → o próprio objeto toma a iniciativa do encerramento

No caso em que houver a comunicação da necessidade de encerramento do serviço por parte do *ViewSharerObject* ao *chairman*, dar-se-á a votação ou a imposição de fechamento. Na figura 7.15, é mostrado um fechamento que ocorre a aprtir do próprio *ContactServiceObject*. O tratamento da política é realiza no *CSCWServices* que evocará, quando necessário um processo de votação, representado pelas mensagens (1) e (2), e caso seja aprovado, dá-se início do fechamento propriamente dito.

Por sua vez, quando o fechamento é requisitado pelo *ViewSharerObject*, ocorrerá de forma semelhante, porém o pedido não mais será feito pelo *ContactService*, cabendo à este apenas a notificação final para que possa eliminar o serviço fechado de sua lista específica.

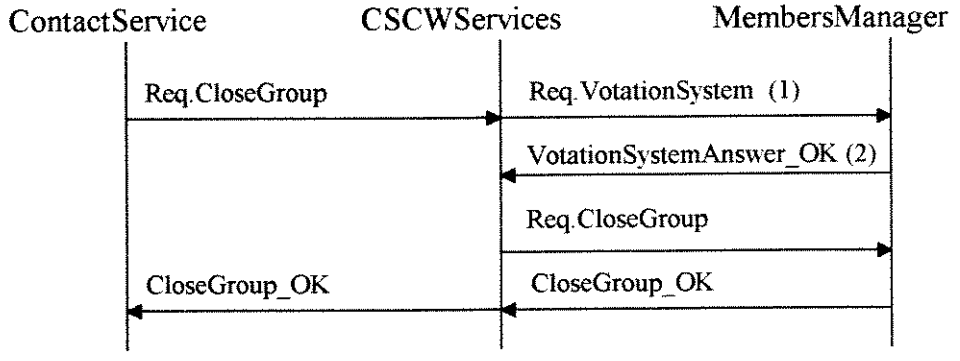


Figura 7.15 - Etapas para o fechamento de um grupo

O encerramento do serviço pelo *ViewSharerObject* quando a qualidade do mesmo cair a níveis inaceitáveis, somente será ocorrido quando o não estabelecimento de uma política que determine simplesmente sua interrupção até que as condições normais sejam restabelecidas dentro de um certo limite de tempo.

A implementação realizada, ocorreu apenas a nível de fechamento pelo próprio chairman de maneira impositiva, ou seja, sem quaisquer consultas aos demais membros ou sem o aviso prévio de tal decisão.

Protótipo:

CloseGroup(IT_env)

Parâmetros:

- *IT_env*: estrutura para verificação de falhas

7.3.3.9 - *CSCWIsochronousService::SetStartTime*

Método acessado na etapa de configuração do serviço para que sejam setadas a data e a hora para o início do cooperativismo.

Caso seus campos possuam valor nulo, o início ficará a cargo do chairman da aplicação.

Protótipo:

SetStartTime(dt, IT_env)

Parâmetros:

- *dt* (const T_DataTime&): data/início para o início do cooperativismo
- *IT_env*: estrutura para a verificação de falhas

7.3.3.10 - *CSCWIsochronousService::SetDurationTime*

Método acessado na etapa de configuração do serviço para que seja setado o tempo de duração do cooperativismo.

Caso seus campos possuam valor nulo, o término ficará a cargo do chairman da aplicação.

Protótipo:

SetDurationTime(t, IT_env)

Parâmetros:

- *t* (const T_Time&): duração do cooperativismo
- *IT_env*: estrutura para a verificação de falhas

7.3.3.11 - *CSCWServiceWithViewSharer::AppendViewSharer*

Na construção de um serviço que envolva a confecção e a instanciação de um objeto do tipo *ViewSharer*, haverá a necessidade de setar os atributos, à ele referentes, presentes na especialização *CSCWServicesWithViewSharer*.

O presente método apenas recebe a informação do *ContactServiceObject* e a copia no referido atributo.

Protótipo:

AppendViewSharer(view, IT_env)

Parâmetros:

- *view* (const T_GeneralObject&): definição do objeto *ViewSharer*
- *IT_env*: estrutura *Orbix* utilizada para o tratamento de falhas

7.3.4 - MembersManager

7.3.4.1 - *MembersManager::ConnectMembers*

A configuração de um membro participante é algo essencial em toda aplicação que manipula cooperativismo, resguardando, assim,as características básicas do mesmo.

Sob o ponto de vista de anexação de membros, encontram-se duas ocasiões totalmente distintas: aquela que o membro faz parte da relação inicial prescrita pelo chairman do grupo e aquela quando o membro deseja incorporar-se a um grupo que já se encontra em execução. Em ambos os casos, o processo é realizado pelo método *ConnectMember1*, que não faz parte da interface exportada pelo objeto, tratando-se de uma rotina básica do mesmo.

O presente método somente irá realizar chamadas ao *ConnectMember1*, para que possam ser criados, individualmente, todos os objetos representativos dos membros, assim como realizar a alocação dos recursos específicos.

A diferença básica deste método com o *ConnectMember*, em relação à origem da evocação, reside no fato de que o *ConnectMembers* é requisitado quando na configuração inicial do grupo, respeitando a relação passada pelo próprio chairman, enquanto que o *ConnectMember* manipula a entrada de membros externos em meio à execução do grupo.

Para melhor ilustrar a situação, supõe-se que todo o processo é feito internamente ao corrente método. Como o pedido de anexação faz parte da configuração inicial do grupo, não serão tratadas as políticas de anexação, processo esse que será tratado na descrição do próximo método.

A figura 7.16 ilustra o processo de configuração inicial dos membros, onde por questões de simplicação, é mostrada a ação apenas sobre um membro, os demais são apenas repetições do mesmo mecanismo.

As mensagens denotadas por (1) e (2) são dependentes do tipo de gerenciador de recursos que será utilizado. O método *AppendToNewResourcesManager* será utilizado quando o membro a ser configurado é o próprio chairman ou o serviço requerer recursos locais. Porém caso o membro sob a fase de configuração consistir em um *NormalMember* e o serviço utiliza recursos compartilhados, o método requisitado será o *AppendToResources-Manager*, anexando o membro criado ao *ResourcesManager* instanciado no momento da configuração do objeto representativo do *chairman*.

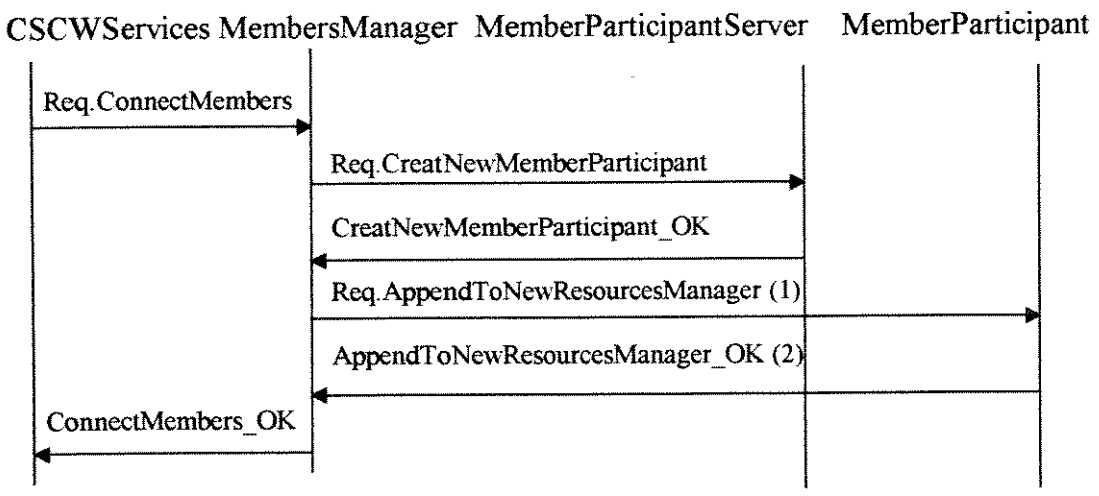


Figura 7.16 - Configuração inicial de um membro participante

Protótipo:

ConnectMembers(members, nummembers, servicename, resmanID, IT_env)

Parâmetros:

- *members* (const _IDL_SEQUENCE_T_Member&): relação de membros a serem anexados
- *nummembers* (short): número de membros para anexação
- *servicename* (const char *): nome do grupo em questão
- *resmanID* (const _IDL_SEQUENCE_T_ResourcesManager&): lista de descritores *ResourcesManager/Resources*
- *IT_env*: estrutura para tratamento de falhas

7.3.4.2 - *MembersManager::ConnectMember*

A inclusão de membros em tempo de execução requer cuidados especiais. Tais cuidados são tratados pelas políticas presentes no vetor de políticas do *MembersManager* (*Policies*).

Para melhor ilustrar o mecanismo desse tipo de anexação, fazer-se-á embutir o método *ConnectMember1* no método *ConnectMember*.

Conforme explicado no item anterior, o método *ConnectMember1* é o real responsável pela anexação dos membros, incluindo a verificação de políticas pertinentes ao processo.

Será suposto que a única política presente exija unicamente a autorização pelo chairman. Neste caso, o método irá enviar uma mensagem do tipo questionamento e aguardar a resposta (permitindo ou não a anexação). Durante o período de aguardo da resposta ou em caso de resposta negativa, o membro solicitante da anexação ficará impedido de incorporar-se ao grupo.

Este mecanismo, portanto consiste de 4 etapas, conforme são ilustradas nas figuras de 7.17 a 7.20.

A figura 7.17 representa o início de um pedido de anexação, cuja política que a trata garante que os pedidos sejam enviados ao chairman sob a forma de questionamento. A mensagem enviada ao chairman consiste em informar que um certo membro está solicitando a sua entrada, conforme é exemplificado:

membername_ADD_EXTERNAL_MEMBER

Ainda na figura 7.17 visualiza-se que o método não retém o controle do membro externo quando na espera da resposta pelo chairman, liberando-o para quaisquer outras atividades.

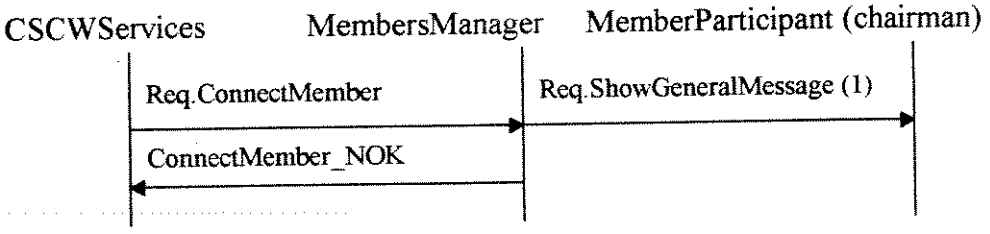


Figura 7.17 - Pedido de anexação gerando um sistema de consulta ao chairman

Por sua vez, a figura 7.18 ilustra o momento em que o membro externo requisita novamente a sua entrada porém a resposta ao questionamento enviado ao chairman ou foi negativa ou ainda não foi recebida.

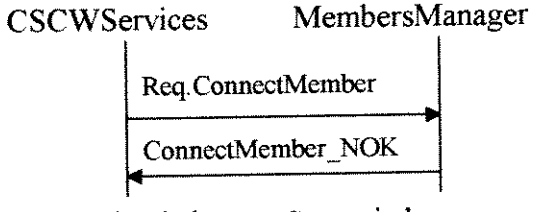


Figura 7.18 - Resposta do chairman não enviada ou negativa

O processo de resposta é realizado pelo método *AnswerQuestion*, conforme ilustra a figura 7.19. A partir desse momento o usuário é liberado a entrar no grupo ou definitivamente ficará sem a sua anexação.

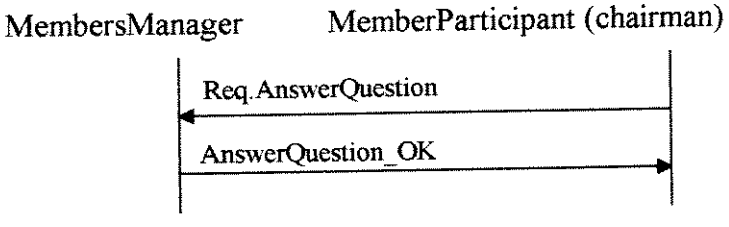


Figura 7.19 - Resposta enviada a partir do chairman do grupo

Caso a resposta seja afirmativa, o processo de anexação é ativado como ilustra a figura de número 7.20.

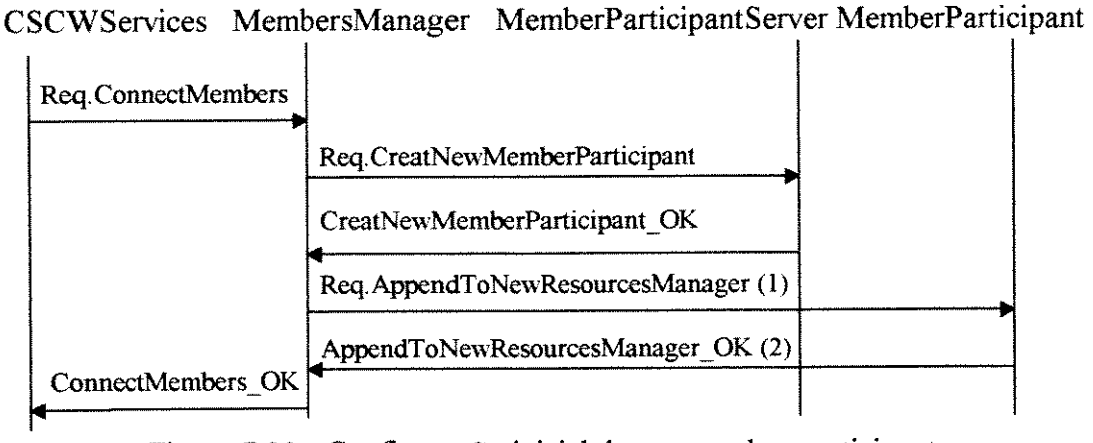


Figura 7.20 - Configuração inicial de um membro participante

Protótipo:

ConnectMember(member, servicename, resmanID, IT_env)

Parâmetros:

- *member* (const T_Member&): membro a ser anexado
- *servicename* (const char *): nome do grupo em questão
- *resmanID* (const _T_ResourcesManager&): informação sobre *Resources-Manager/Resources*
- *IT_env*: estrutura para tratamento de falhas

7.3.4.3 - *MembersManagerXXX::InitXMember*

Os membros, após sua etapa de configuração, encontram-se no estado *NOACTIVE*, permitindo uma flexibilidade em relação ao estado corrente do grupo em relação aos seus membros.

A passagem de *NOACTIVE* para *ACTIVE* consiste na efetiva abertura dos canais e conexões dos recursos alocados ao mesmo, assim como uma notificação, através do canal de mensagens gerais, aos membros já ativos.

O mecanismo de ativação de um membro é mostrado na figura 7.21. O método *InitXMember* (onde o 'X' substitui as letras 'I', 'A' ou 'V', diferenciando os métodos pertencentes às especializações do objeto MembersManager: MembersManagerIsochronous, *MembersManagerAnisochronous* e *MembersManagerWithViewSharer*) engloba a chamada ao método não pertencente à interface: *InitMember1*.

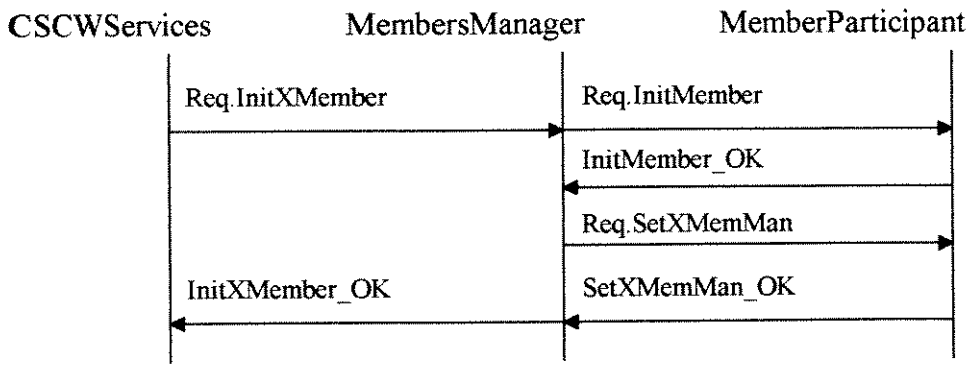


Figura 7.21 - Ativação de um membro

Protótipo:

InitMember(member, IT_env)

Parâmetros:

- *member* (const char *): nome do membro a ser ativado
- *IT_env*: estrutura *Orbix* para tratamento de falhas

7.3.4.4 - *MembersManager::ActiveIGroup* *MembersManager::ActiveAGroup* *MembersManager::ActiveVGroup*

O método *ActiveXGroup* apenas realiza a instanciação do ponteiro referente ao objeto *CSCWServices* (figura 7.22).

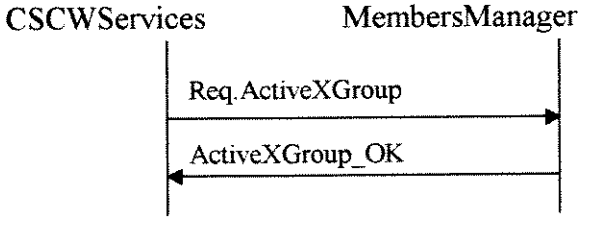


Figura 7.22 - Instanciação do atributo *CSCW* do *MembersManager*

Protótipo:

ActiveIGroup(cscw,IT_env)
ActiveAGroup(cscw,IT_env)
ActiveVGroup(cscw,IT_env)

Parâmetros:

- *cscw*: ponteiro para *CSCWServices* (tipos abrangidos: CSCWIsochronous-Service, CSCWAnisochronousService ou CSCWServiceWithViewSharer)
- *IT_env*: utilizada para detecção de falha de execução

7.3.4.5 - *MembersManager::CloseGroup*

Seguindo a filosofia de dependência hierárquica em relação à evocação de métodos com finalidades em comum, encontra-se, no *MembersManager*, um método que complementa o método de mesmo nome do *CSCWServices*.

O real fechamento é feito em pós-ordem, ou seja, somente irá ser processado naquele nível caso os níveis inferiores tenham sido fechados corretamente, conforme é mostrado na figura 7.23.

Os métodos denotados pela numeração (1) a (4) são repetidos a cada membro configurado no serviço, para que possa realizar o fechamento das conexões envolvidas a cada membro de forma individual.

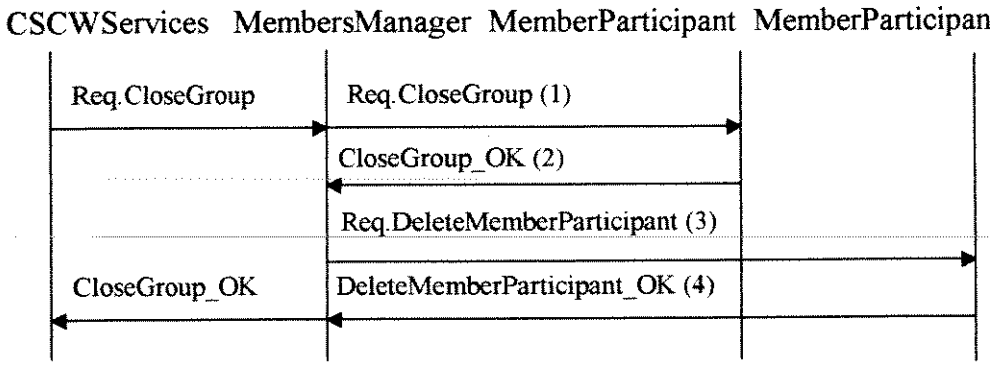


Figura 7.23 - Fechamento de um grupo a nível do *MembersManager*

Protótipo:

CloseGroup(IT_env)

Parâmetros:

- *IT_env*: estrutura para o tratamento de falhas

7.3.4.6 - *MembersManager::GetInfoMember*

Uma sociedade, em qualquer forma de representação, requer a identificação pessoal de todos os membros que dela participam. Em relação à camada Groupware, a forma de obtenção de tal informação é através do presente método, o qual, a partir do nome do membro, devolve a sua localização, o papel e o tipo do objeto de abstração.

Protótipo:

GetInfoMember(member,host,role,type,IT_env)

Parâmetros:

- *member* (const char *): nome do membro a ser consultado
- *host* (char *&): localização (host) do membro consultado

- *role* (char *&): papel desempenhado por tal membro
- *type* (short&): tipo do objeto que o representa
- *IT_env*: estrutura para detecção de falhas

7.3.4.7 - *MembersManager::AnswerQuestion*

A necessidade de questionamento para a tomada de autorizações, tanto de ordem técnica quanto relacionadas com o próprio tema do serviço, faz com que seja útil a introdução de um método que realize todo o processo de recolhimento de respostas.

Na verdade, o membro que faz uso desse método está respondendo a uma questão comunicada por meio do canal de mensagens gerais do grupo e armazenada em um lista de questões pendentes.

A resposta recolhida irá dar continuação ao tratamento da política que exige a intervenção de algum membro, como por exemplo, a do chairman.

Na implementação final, este método é evocado na confirmação, positiva ou não, do pedido de anexação realizado por um membro externo ao grupo.

O método é bidirecional, enviando como resultado a interpretação correta da resposta (7.24).

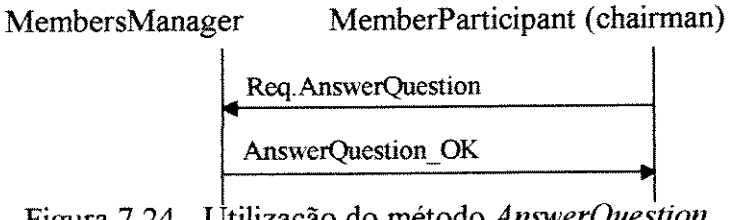


Figura 7.24 - Utilização do método *AnswerQuestion*

Protótipo:

AnswerQuestion(question,value,IT_env)

Parâmetros:

- *question* (const char *): descrição da questão a ser respondida
- *value* (short): valor da resposta
- *IT_env*: estrutura para detecção de falhas

7.3.4.8 - *MembersManagerIsochronous::SendGeneralMessage*

Um serviço cooperativo exige troca de informações entre os seus membros. Podemos considerar três tipos de mensagens: formal, informal e a terceira como mensagens da própria camada ou da aplicação cooperativa em execução. A formal é aquela originada pelo membro que detém o canal de comunicação, onde tal, possuidor do direito de transmitir naquele canal, o utiliza enviando sua mensagem a todos os demais membros. A informal é aquela feita entre dois membros

quaisquer, sem distinção de papel e de direitos e, por último, as mensagens do sistema originam-se do próprio andamento do cooperativismo.

A distribuição das mensagens formais e de mensagens gerais é feita sob a responsabilidade do *MembersManagerObject* que, por meio de sistema de *broadcast*, consegue o objetivo de que todos os demais membros recebam a mensagem, enquanto que as mensagens informais são tratadas diretamente pelos membros participantes envolvidos.

Tal distribuição realizada pelo método *SendGeneralMessage* visa a utilização de canais de mensagens gerais, já que o *broadcast* pelo canal formal é alcançado pelo método *SendFormalMessage*.

Protótipo:

SendGeneralMessage(message)

Parâmetros:

- *message* (const char *): mensagem a ser distribuída
- *IT_env*: estrutura para tratamento de falhas

7.3.4.9 - *MembersManagerIsochronous::SendFormalMessage*

Responsável pelo envio das mensagens formais, identificando, antes enviá-las, o detentor do canal formal de comunicação para que possa, se possível, distribuí-las para todos os membros ativos de um serviço isócrono (7.25).

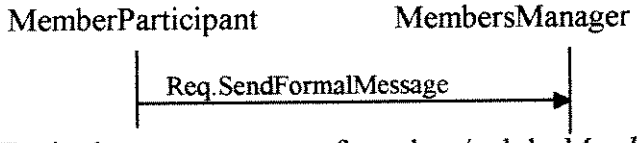


Figura 7.25 - Envio de uma mensagem formal a nível do *MemberParticipant*

Protótipo:

SendFormalMessage(message,smember,shost,IT_env)

Parâmetros:

- *message* (const char *): mensagem a ser distribuída
- *smember* (const char *): nome do membro origem
- *shost* (const char *): host do membro origem
- *IT_env*: estrutura para o tratamento de falhas

7.3.4.10 - *MembersManagerIsochronous::ChangeCurrentMember*

A manipulação do canal formal de comunicação requer uma dinamicidade quanto ao membro que o utiliza no momento. Para tanto, faz-se necessário um método que atue na substituição do membro corrente, permitindo que várias

pessoas possam utilizá-lo. Apenas uma única pessoa é detentora da permissão de uso em um dado momento.

A troca de membro corrente segue uma das seguintes políticas:

- determinado por um tempo de utilização, sendo substituído pelo primeiro membro que realizou uma inscrição para a manipulação
- determinado por um tempo de utilização, sendo substituído por um membro indicado pelo *chairman*
- não tendo tempo específico, a substituição é realizada pelo *chairman* aleatoriamente

A figura 7.26 mostra a substituição ordenada pela ação direta do *chairman*, sem a contagem de tempo máximo de utilização do canal formal.

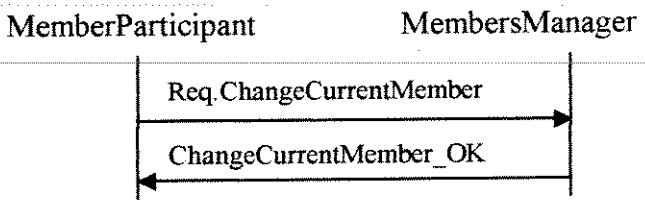


Figura 7.26 - Mudança do membro corrente

Protótipo:

ChangeCurrenteMember(newname,newhost,IT_env)

Parâmetros:

- *newname* (const char *): nome do membro que assumirá o direito de utilização do canal formal de comunicação
- *newhost* (const char *): host relacionado ao novo membro
- *IT_env*: estrutura *Orbix* para checagem de falhas

7.3.4.11 - *MembersManagerIsochronous::InterruptMember*

A finalização da ação local representa a saída do membro em relação ao grupo cooperativo no qual está inserido. Para tanto, o processo se iniciará com o fechamento dos recursos e dos canais associados ao membro em questão para que possa, então, efetivar o seu desligamento, culminando com o envio de um aviso aos restantes por intermédio do canal de informações gerais.

As primeiras etapas serão manipuladas pelo objeto *MemberParticipant*, conforme será descrito oportunamente. A participação do *MembersManager* neste processo reside apenas em colocação de um valor nulo no vetor de ponteiros para as interfaces dos membros (*MemPar*) e, em seguida, realizar o *broadcast* avisando o novo estado do membro: "NOACTIVE".

Protótipo:

InterruptMember(name,host,IT_env)

Parâmetros:

- *name* (const char *): nome do membro a ser fechado
- *host* (const char *): localização do membro
- *IT_env*: estrutura para manipulação de falhas

7.3.5 - MemberParticipant

7.3.5.1 - *MemberParticipant::AppendToResourcesManager*

Em serviços cooperativos onde a manipulação de recursos é feita compartilhadamente (figura 7.27), a etapa de construção de um membro não é seguida pela instanciação de um novo *ResourcesManager*, mas sim pela conexão a um já criado.

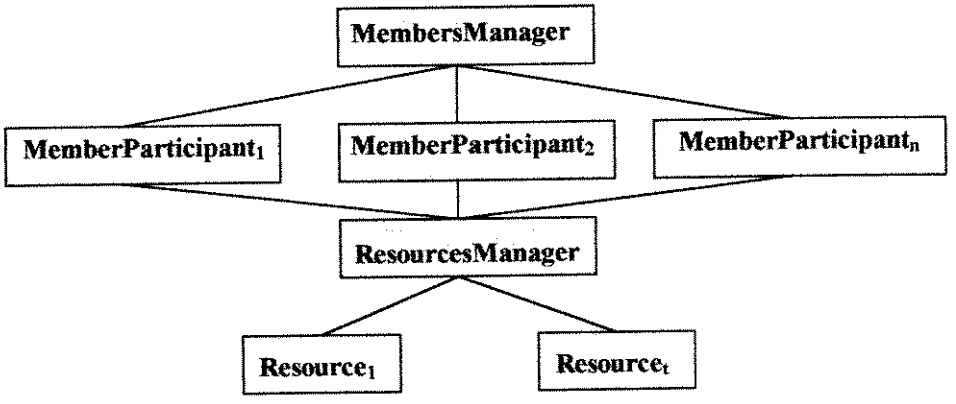


Figura 7.27 - Serviço cooperativo com recursos compartilhados

Neste caso, a criação de um *ResourcesManager* é realizada na instanciação do primeiro membro presente na lista de membros participantes, sendo este representado pela figura do *chairman*. A anexação do *chairman*, ou de qualquer membro em um serviço com recursos locais, ao *ResourcesManager* é conseguido pelo método *AppendToNewResources-Manager*, descrito no próximo item.

A figura 7.28 mostra o intercâmbio de mensagens entre os objetos para que seja alcançada a meta de anexação de um membro a um gerenciador de recursos.

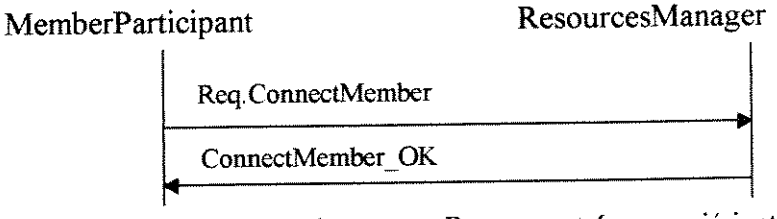


Figura 7.28 - Anexação do membro a um *ResourcesManager* já instanciado

Para a execução do método, a mensagem emitida pelo membro engloba a sua própria identificação, representada pelo atributo *MemberID*. O ponteiro para o *ResourcesManager* é passada ao *MemberParticipant* pelo objeto *MembersManager*, o qual neste momento gerencia a etapa de instanciação dos membros.

Protótipo:

AppendToResourcesManager(resman,IT_env)

Parâmetros:

- *resman* (const T_GeneralObject&): ponteiro para o *ResourcesManager* que receberá a conexão do membro
- *IT_env*: estrutura utilizada para a checagem de falhas de comunicação

7.3.5.2 - *MemberParticipant::AppendToNewResourcesManager*

Ao contrário do método *AppendToResourcesManager*, o presente método irá atuar nos serviços cujos recursos serão alocados localmente para cada usuário (figura 7.29). Neste caso, haverá a instanciação de um *ResourcesManager* para cada membro, onde o diagrama de troca de mensagens é ilustrado na figura 7.30.

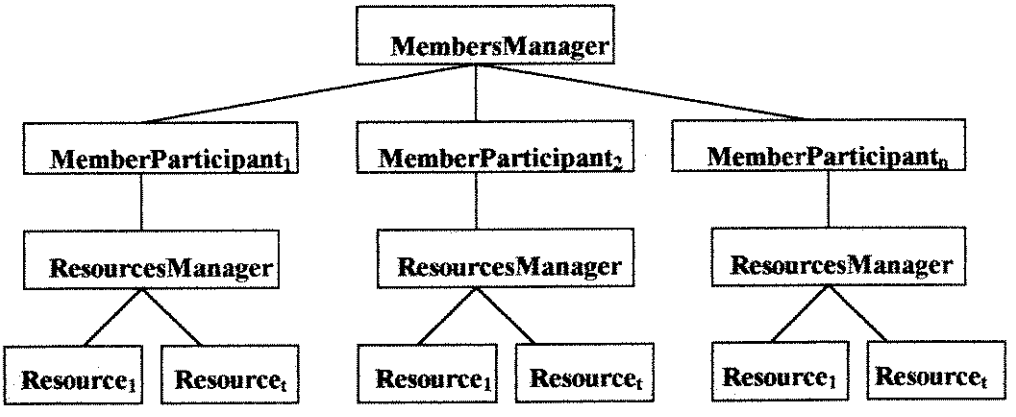


Figura 7.29 - Serviço cooperativo com recursos locais

Este método é evocado na ocasião da instanciação do *chairman* tanto nos serviços cooperativos com recursos locais quanto aqueles com recursos compartilhados. No último caso, o parâmetro *resman* é retornado ao *MembersManager* para que possa chamar corretamente o método *AppendToResourcesManager* conforme é descrito no item anterior.

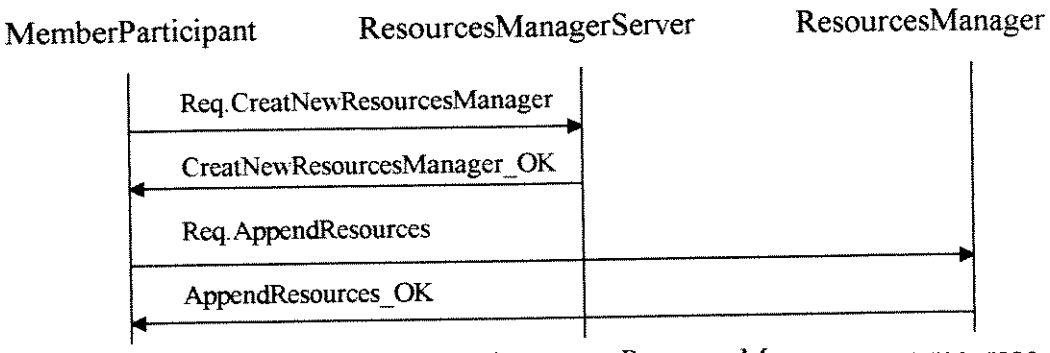


Figura 7.30 - Anexação de um membro a um *ResourceManager* com recursos locais

Protótipo:

AppendToNewResourcesManager(servicename,resmanID,resman,IT_env)

Parâmetros:

- *servicename* (const char *): nome do serviço cooperativo corrente
- *resmanID* (const _T_ResourcesManager&): definição do *ResourcesManager* a ser instanciado
- *resman* (T_GeneralObject&): identificação do *ResourcesManager* recém criado
- *IT_env*: estrutura para verificação de falhas de comunicação

7.3.5.3 - *Memberparticipant::SetIMemMan/SetAMemMan/SetVMemMan*

Na ocasião do início das operações locais de cada membro participante, os referidos métodos são executados para instanciar o atributo *MemMan*, o qual realiza a referência para o objeto *MembersManager*.

A diferença entre os métodos *SetIMemMan*, *SetAMemMan* e *SetVMemMan* se baseia no tipo de *MembersManager* abrangido: *MembersManagerIsochronous*, *MembersManagerAnisochronous* e *MembersManagerWithViewSharer*, respectivamente.

Protótipo:

SetIMemMan(IT_env)
SetAMemMan(IT_env)
SetVMemMan(IT_env)
Parâmetros:

- *IT_env*: estrutura para verificação de falhas de comunicação

7.3.5.4 - *MemberParticipant::GetStatus*

Tratando-se de ações cooperativas, as quais se fundamentam no intercâmbio de informações entre os seus membros participantes, é importante a obtenção do estado destes, diferenciando-se como *ACTIVE* e *NOACTIVE*.

A obtenção de tal atributo é conseguida pelo método *GetStatus* por intermédio de seu retorno.

Protótipo:

GetStatus(IT_env)

Parâmetros

- *IT_env* → estrutura para checagem de falhas de comunicação
- retorno da função (short): Estado corrente do membro

7.3.5.5 -*MemberParticipant::SetStatus*

Conforme já abordado, o estado de um membro o caracteriza quanto à sua participação em relação ao grupo, podendo estar sob a forma ativa e inativa. A transição dos estados é conseguida pela evocação do método *SetStatus*, onde o seu novo estado corresponde ao valor passado como parâmetro.

Protótipo:

SetStatus(value,IT_env)

Parâmetros:

- *value* (short): novo estado a ser assumido pelo membro
- *IT_env*: estrutura para checagem de falhas de comunicação

7.3.5.6 -*MemberParticipant::OpenDInformalChannel*

Sendo os canais informais utilizados para a comunicação direta entre dois membros quaisquer dentro de um grupo cooperativo, a sua abertura requer duas etapas: a primeira consiste na abertura dos recursos necessários no membro destino e, a segunda, na abertura dos recursos locais ao membro requisitante do canal, assegurando assim a abertura bilateral.

A primeira etapa é conseguida pelo método *OpenDInformalChannel*, evocado remotamente pelo requisitante por intermédio do *OpenSInformalChannel*.

O presente método realizará o intercâmbio de mensagens entre o *ResourcesManager*, instanciando em seguida os atributos relacionados com o vetor destinado aos canais informais (*InformalChannelID*), incluindo a referência do próprio canal (*InfoChannels*), obtida após a evocação do método *_bind* da camada de gerenciamento dos objetos distribuídos (figura 7.31).

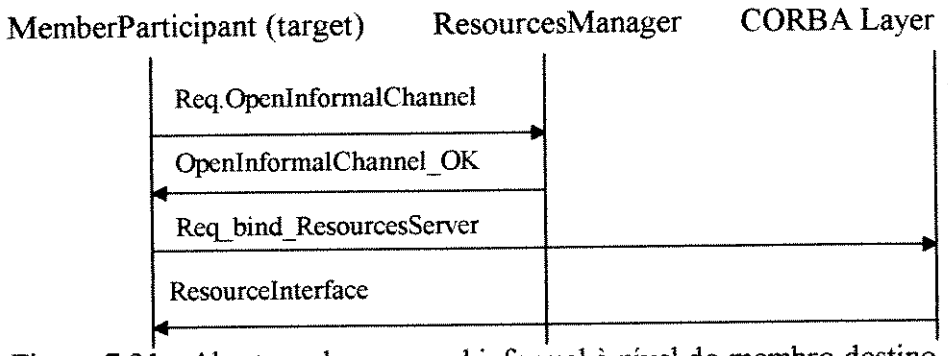


Figura 7.31 - Abertura de um canal informal à nível do membro destino

Protótipo:

OpenDInformalChannel(type,smember,shost,dmember,IT_env)

Parâmetros:

- *type* (short): tipo do membro requerente da abertura (*source*)
- *smember* (const char *): nome do emissor
- *shost* (const char *): localização do membro *source*
- *dmember* (const char *): nome do membro destinatário
- *IT_env*: estrutura para checagem de falhas de comunicação

7.3.5.7 - *MemberParticipant::OpenSInformalChannel*

Conforme abordado na descrição do método *OpenDInformalChannel*, o estabelecimen-to de um canal informal de comunicação abrange duas etapas: a abertura remota dos recursos no membro destino e a abertura local dos recursos.

O gerenciamento destas é feito por intermédio do método *OpenSInformalChannel*, evocado pelo membro interessado em associar-se informalmente a um outro do mesmo grupo cooperativo. A figura 7.32 ilustra a seqüência de criação de um canal informal de comunicação.

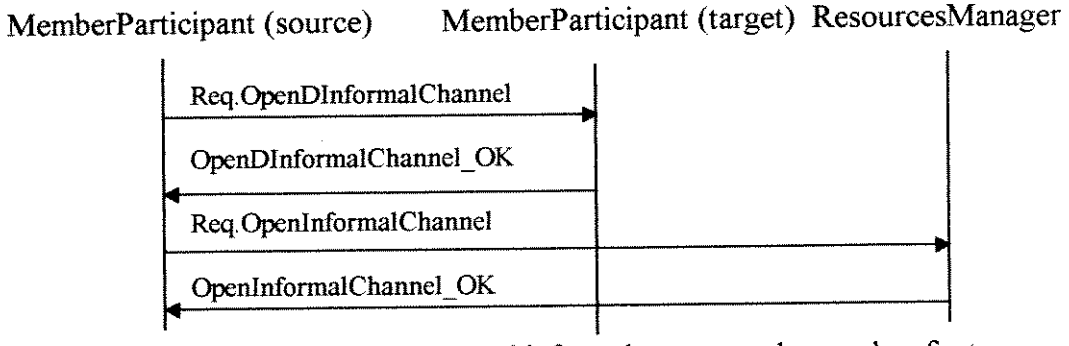


Figura 7.32 - Abertura de um canal informal por parte do membro fonte

Protótipo:

OpenSInformalChannel(TypeOfResource,TargetMember,SourceMember,
SourceHost,IT_X)

Parâmetros:

- TypeOfResource (short): Tipo de recurso associado ao canal de comunicação. Atualmente a plataforma suporta apenas janelas de texto do tipo "*X-Window*" (TEXTUAL_VIDEO). Parâmetro de entrada.
- TargetMember (const char *): Nome do membro que será associado ao canal informal. Parâmetro de entrada.
- SourceMember (const char *): Nome do membro que está requisitando a abertura do canal. Parâmetro de entrada.
- SourceHost (const char *): Localização do membro solicitante da abertura. Parâmetro de entrada.
- *IT_env* (IT_X): Estrutura para checagem de falhas de comunicação
- Retorno da função (short): Indicativo de erro.

7.3.5.8 - *MemberParticipant::CloseRDInformalChannel*

Assim como a abertura, o encerramento da conexão de um canal informal compreende a evocação do fechamento dos recursos associados remotamente e o fechamento dos recursos localizados localmente ao membro requisitante do fechamento.

O método correspondente ao fechamento remoto (*CloseRDInformal-Channel*) consiste simplesmente na destruição da referência do canal armazenado no vetor *InformalChannelID*, onde tal método é chamado pelo membro requisitante do fechamento do canal por intermédio do método *CloseInformal-Channel*, conforme será abordado no próximo item.

Protótipo:

CloseRDInformalChannel(chname,IT_env)

Parâmetros:

- *chname* (const char *): nome do canal, corresponde ao nome do membro associado informalmente
- *IT_env*: estrutura para verificação de falhas de comunicação.

7.3.5.9 - *MemberParticipant::CloseInformalChannel*

O completo gerenciamento para o fechamento de um canal informal de comunicação é realizado pelo método *CloseInformalChannel*, evocado a partir do membro que requisita o fechamento do mesmo.

Para procedê-lo, tal método primeiramente efetua o fechamento dos recursos abertos remotamente (membro associado informalmente) e, em seguida, os abertos localmente à máquina do membro solicitante, conforme ilustra a figura 7.33.

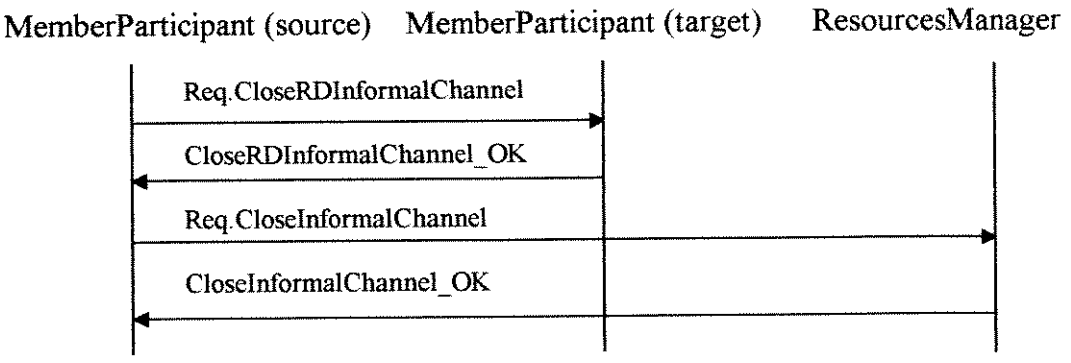


Figura 7.33 - Fechamento de um canal informal à nível do requisitante

Protótipo:

CloseInformalChannel(TargetMember,SourceMember,SourceHost,IT_X)

Parâmetros:

- TargetMember (const char *): Nome do membro destinatário associado ao canal. Parâmetro de entrada.
- SourceMember (const char *): Membro solicitante do fechamento. Parâmetro de entrada.
- SourceHost (const char *): Localização (host) do membro requisitante. Parâmetro de entrada.
- *IT_env*: Estrutura para verificação de falhas de comunicação
- Retorno da função (short): indicativo de erro.

7.3.5.10 - *MemberParticipant::Interrupt*

A interrupção de um membro corresponde a uma saída momentânea, ou seja, o membro poderá incorporar-se novamente pela evocação do método *InitMember* pertencente ao objeto *ContactService*.

Pela razão da momentaneidade da saída, o método *Interrupt* realiza o apenas o fechamento dos recursos, deixando suas descrições nas devidas estruturas.

O processo de interrupção de um membro é ilustrado pela figura 7.34.

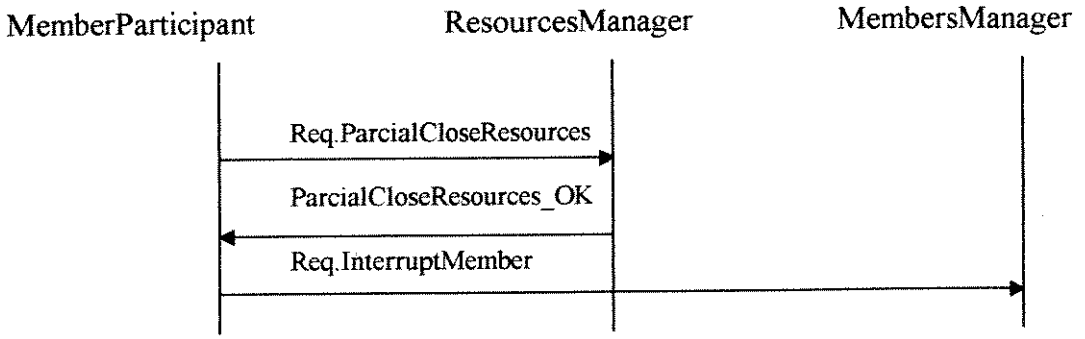


Figura 7.34 - Interrupção de um membro

Além dos métodos presentes no figura acima, a interrupção de um membro engloba o encerramento de todos os canais informais de comunicação, abertos no momento, à ele associados.

Protótipo:

Interrupt(LoginName,Host,IT_X)

Parâmetros:

- LoginName (const char *): Nome do membro solicitante da interrupção. Parâmetro de entrada.
- Host (const char *): Localização do requisitante (host). Parâmetro de entrada.
- *IT_env*: Estrutura para detecção de falhas de comunicação
- Retorno da função (short): Indicativo de erro.

7.3.5.11 - *MemberParticipant::SendFormalMessage*

A permissão de utilização do canal de mensagens formais é checada a nível do objeto *MembersManager*, portanto o método *SendFormalMessage* apenas tem a função de evocar o método homônimo pertencente àquele objeto (figura 7.35).

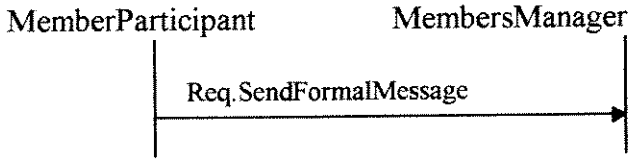


Figura 7.35 - Envio de uma mensagem formal

Protótipo:

SendFormalMessage(Message,IT_X)

Parâmetros:

- Message (const char *): Mensagem a ser enviada pelo canal formal de comunicação. Parâmetro de entrada.
- *IT_env*: Estrutura para chacagem de falhas de comunicação

7.3.5.12 - *MemberParticipant::SendInformalMessage*

A diferença no envio de mensagens formais e informais consiste, além da verificação da permissibilidade, na identificação do objeto que receberá a mensagem: enquanto que na comunicação formal, o objeto responsável pela distribuição da mensagem via *broadcast* é representado pelo *MembersManager*; na comunicação informal, como a conexão visa a interligação entre dois membros do

grupo, a mesagem é diretamente enviada ao recurso (janela) remoto do destinatário (figura 7.36).

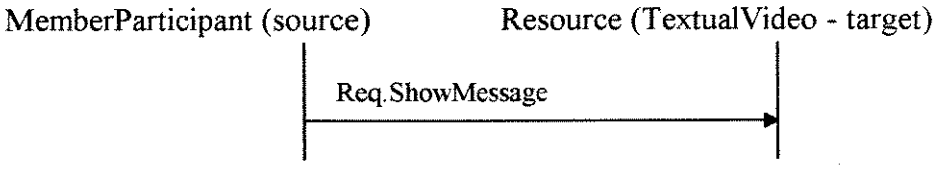


Figura 7.36 - Envio de uma mensagem informal

Protótipo:

SendInformalMessage(Message,TargetMember,IT_X)

Parâmetros:

- Message (const char *): Mensagem a ser enviada. Parâmetro de entrada.
- TargetMember (const char*): Membro destinatário da mensagem. Parâmetro de entrada.
- *IT_env*: Estrutura de checagem de falhas de comunicação

7.3.5.13 - *MemberParticipant::SendGeneralMessage*

O envio de uma mensagem genérica é realizado sem a intervenção direta do membro participante, pois representa mensagens originadas pela camada *Groupware*, ou pela aplicação cooperativa em execução, para comunicações gerais ou pertencentes, por exemplo, aos mecanismos de questionamento.

Para tanto, apenas envia a mensagem ao *MembersManager* que realizará um *broadcast* conforme ilustra a figura 7.37.

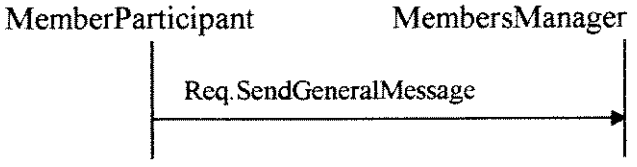


Figura 7.37 - Envio de uma mensagem genérica

Protótipo:

SendGeneralMessage(Message,IT_X)

Parâmetros:

- Message (const char *): Mensagem a ser enviada pelo canal geral de comunicação. Parâmetro de entrada.
- *IT_env*: Estrutura para detecção de falhas de comunicação

7.3.5.14 - *MemberParticipant::ShowFormalMessage*

O método de exibição de uma mensagem formal consiste apenas na sua retransmissão ao objeto *ResourcesManager* (figura 7.38).

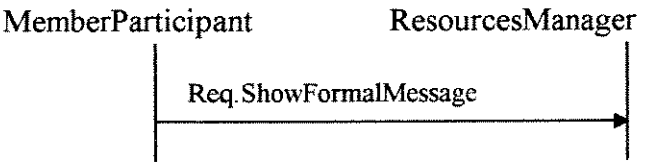


Figura 7.38 - Exibição de uma mensagem formal

Protótipo:

ShowFormalMessage(message,smember,shost,IT_env)

Parâmetros:

- *message* (const char *): mesagem a ser exibida
- *smember* (const char *): nome do membro emissor da mensagem (membro corrente)
- *shost* (const char *): localização do membro emissor da mensagem
- *IT_env*: estrutura para verificação de falhas de comunicação

7.3.5.15 - *MemberParticipant::ShowGeneralMessage*

O método de exibição de uma mensagem genérica (mensagens da camada) consiste apenas na retransmissão da mesma ao objeto *ResourcesManager* (figura 7.39).

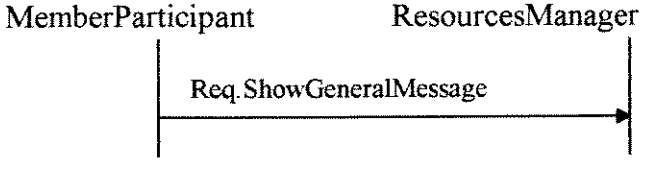


Figura 7.39 - Exibição de uma mensagem genérica

Protótipo:

ShowGeneralMessage(message,IT_env)

Parâmetros:

- *message* (const char *): mensagem a ser exibida
- *IT_env*: estrutura para detecção de falhas de comunicação

7.3.5.16 - *MemberParticipant::GetMemberName*

A facilidade de identificação dos membros pertencentes a um certo grupo cooperativo é assegurada pelo método *GetMemberName*, o qual consiste na devolução da descrição do membro: nome (retorno da própria função), host, papel dentro do grupo e tipo (*chairman* ou *normal*).

Protótipo:

GetMemberName(host,role,type,IT_env)

Parâmetros:

- *host* (char *&): host do membro sob indagação
- *role* (char *&): papel desempenhado pelo membro
- *type* (short&): tipo do membro (chairman, normal_member)
- *IT_env*: estrutura para detecção de falhas de comunicação

7.3.5.17 - *MemberParticipant::CloseGroup*

O processo de fechamento de um grupo, a nível do *MemberParticipant*, é responsável pelo encerramento das atividades na camadas inferiores, ou seja, apenas enviará uma requisição ao *ResourcesManager* para que sejam fechados todos os recursos, incluindo os canais informais de comunicação, abertos no momento (figura 7.40).

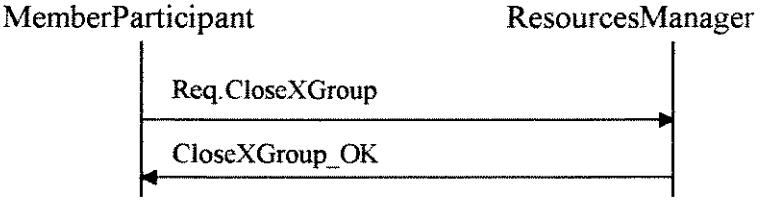


Figura 7.40 - Fechamento do grupo a nível do *MemberParticipant*

Onde poderão ser chamados dois tipos de métodos do *ResourcesManager*:

- *CloseLGroup*: implementado para recursos locais
- *CloseSGroup*: implementado para recursos compartilhados

Após o completo fechamento por parte dos gerenciadores de recursos, o método *CloseGroup* realiza uma chamada para a plataforma de objetos distribuídos, de modo a destruir as instanciações dos objetos *ResourcesManager* (método *_release*).

Protótipo:

CloseGroup(IT_env)

Parâmetros:

- *IT_env*: estrutura para detecção de falhas de comunicação da rede
- retorno da função (short): indicativo de erro (0 = ausência)

7.3.5.18 - *MemberParticipant::AnswerQuestion*

A camada *Groupware* abre a possibilidade do intercâmbio de questões para auxiliar, por exemplo, a tomada de decisões referentes às políticas adotadas para o gerenciamento do serviço cooperativo.

Pertencendo ao *MembersManager*, existe uma lista de questões pendentes, a qual será acessada pelo método de envio de respostas do *MemberParticipant* que propagará a mensagem ao método de mesmo nome pertencente ao *MembersManager* (figura 7.41).

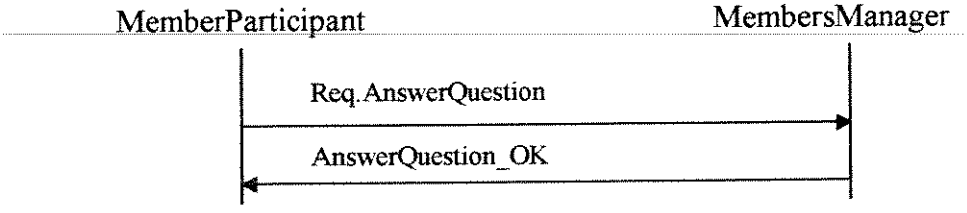


Figura 7.41 - Mecanismo de resposta às questões pendentes

Protótipo:

AnswerQuestion(Question,AnswerValue,IT_X)

Parâmetros:

- Question (const char *): Identificação da questão a ser respondida. Parâmetro de entrada.
- AnswerValue (short): Valor da resposta referente à questão. Parâmetro de entrada.
- *IT_env*: Estrutura para detecção de falhas de comunicação da rede
- Retorno da função (short): Indicativo de erro.

7.3.5.19 - *ChairmanMember::InitCMember/NormalMember::InitNMember*

A etapa de iniciação dos membros, tanto membros do tipo *chairman* quanto do tipo *normal*, corresponde à introdução do membro ao vetor de membros abrangidos pelo *ResourcesManager* (recursos compartilhados) ou à instanciação do ponteiro correspondente ao membro proprietário (recursos locais), neste último caso, também inclui-se a abertura dos recursos alocados.

O processo de iniciação é ilustrada por intermédio da figura 7.42, onde o método *SetXXMemPar* é responsável pela introdução, no *ResourcesManager*, do

ponteiro referente ao próprio membro. Existem variações desse método para que sejam cobertos os diferentes tipos de serviços cooperativos, listados abaixo:

- *SetLCMemPar*: recursos locais e membros do tipo *chairman*
- *SetSCMemPar*: recursos compartilhados e membros do tipo *chairman*
- *SetLNMemPar*: recursos locais e membros do tipo *normal*
- *SetSNMemPar*: recursos compartilhados e membros do tipo *normal*

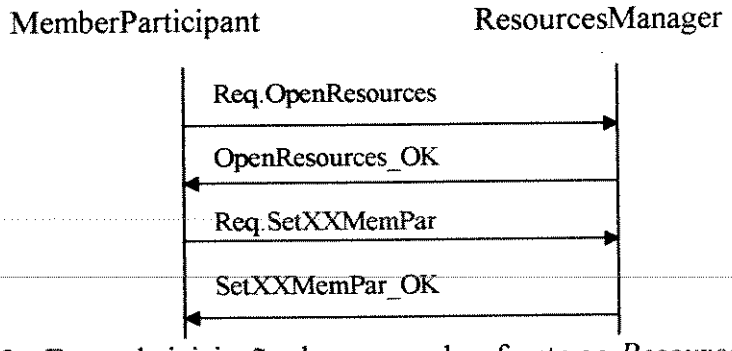


Figura 7.42 - Etapa de iniciação de um membro frente ao *ResourcesManager*

Protótipo:

InitCMember(IT_env)
InitNMember(IT_env)

Parâmetros:

- *IT_env*: ocorrência de falha na rede de comunicação
- retorno do método (short): ocorrência de erro (0 = sem erros)

7.3.5.20 - *ChairmanMember::ChangeCurrentMember*

Ao *chairman* do grupo cooperativo cabe, quando a política adotada permitir, a responsabilidade de escalonamento do membro corrente detentor do direito de utilização do canal formal de comunicação.

Tal funcionalidade é conseguida pela ativação do método *ChangeCurrentMember*, o qual irá propagar a mensagem ao *MembersManager*, afim de realizar efetivamente a troca (figura 7.43).

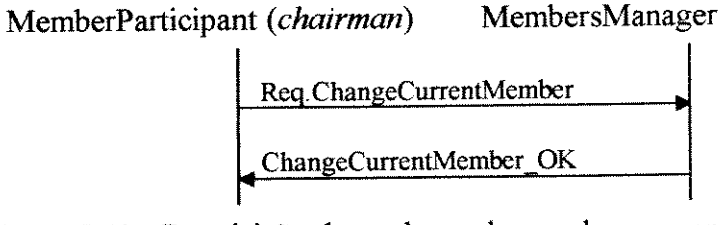


Figura 7.43 - Requisição da mudança do membro corrente

Protótipo:

ChangeCurrentMember(NewMember,IT_X)

Parâmetros:

- NewMember (const char *): Nome do novo membro que assumirá o papel de membro corrente, detentor da utilização do canal formal de comunicação. Parâmetro de entrada.
- *IT_env*: Estrutura para verificação de falhas de comunicação
- Retorno da função (short): Indicativo de erro.

7.3.6 - ResourcesManager

7.3.6.1 - *ResourcesManager::AppendResources*

A última etapa na confecção de um grupo cooperativo consiste no adicionamento dos recursos alocados, através da instanciação de seus objetos e interfaces, diferenciando-os pela qualidade do compartilhamento ou do uso local.

O método *AppendResources* representa essa adição, culminando pelo preenchimento do vetor que armazena as definições dos recursos, representado pelo item *Resources* do *ResourcesManagerID*. A figura 7.44 ilustra o processo de anexação dos recursos.

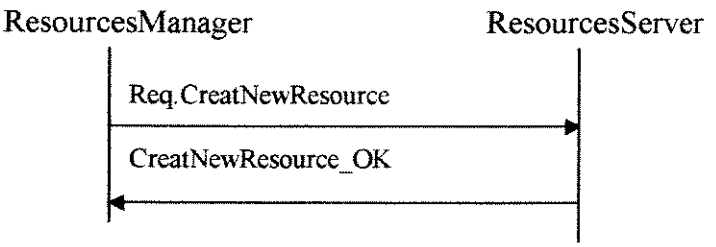


Figura 7.44 - Adição de recursos durante a construção do grupo cooperativo

A seqüência mostrada na figura acima é realizada para todos os recursos presentes na lista *resourcesid* passada ao método como parâmetro.

Protótipo:

AppendResources(minqos,numresources,resourcesid,IT_env)

Parâmetros:

- *minqos* (short): qualidade de serviço esperada para o *ResourcesManager*
- *numresources* (short): número de recursos a serem anexados

- *resourcesid* (const _IDL_SEQUENCE_T_Resource&): descrição dos referidos recursos
- *IT_env*: estrutura para detecção de falhas de comunicação
- retorno da função (short): estado de execução (0=não ocorrência de falhas)

7.3.6.2 - *ResourcesManager::OpenResources*

Para o início de uma ação cooperativa são necessárias duas etapas: a primeira consiste na confecção do grupo, representada pelo processo de configuração de sua arquitetura; a segunda visa o início das ações temporais e sociais do cooperativismo.

Em relação aos recursos, a confecção é abrangida pelo método *AppendResources* e a sua real abertura é alcançada pela ativação do *OpenResources*, o qual será responsável pela evocação direta dos métodos de abertura pertencentes aos recursos, como por exemplo o *OpenDisplay* do objeto de recurso *TextualVideo*.

Nesta etapa ocorre a instanciação da lista *Resources* a qual contém os ponteiros que referenciam as interfaces dos recursos abertos (figura 7.45).

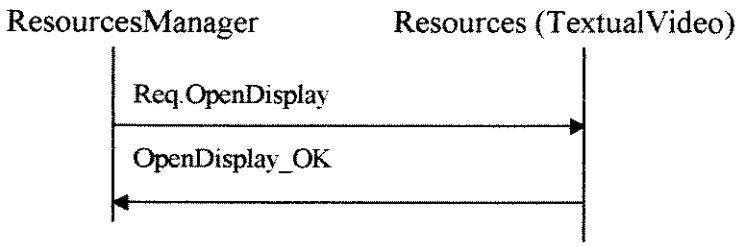


Figura 7.45 - Abertura dos recursos para a iniciação do cooperativismo

Protótipo:

OpenResources(IT_env)

Parâmetros:

- *IT_env* → estrutura para a checagem de falhas de comunicação
- retorno da função (short): indicativo de erro (0 = ausência)

7.3.6.3 - *ResourcesManager::OpenInformalChannel*

Na camada *Groupware*, os canais informais de comunicação são tratados como recursos comuns interligando dois membros quaisquer do grupo. Sendo assim o presente método simplesmente requisitará ao *ResourcesServer* a abertura de um novo recurso, sendo que o tratamento e o armazenamento de suas descrições ficarão a cargo dos membros envolvidos. Adota-se como nome do recurso em questão, a identificação do membro associado através do canal informal. A figura 7.46 representa o presente método.

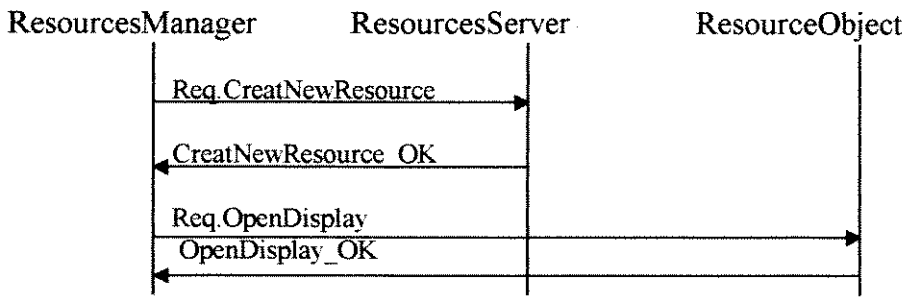


Figura 7.46 - Aberura de um canal informal de comunicação

Protótipo:

OpenInformalChannel(type,dmember,IT_env)

Parâmetros:

- *type* (short): tipo do recurso a ser aberto
- *dmember* (const char *): nome do membro remotamente associado
- *IT_env*: estrutura para a manipulação de falhas de comunicação
- retorno do método (short): indicativo de erro de execução (0=ausência)

7.3.6.4 - *ResourcesManager::CloseInformalChannel*

O tratamento, a nível do *ResourcesManager*, para o fechamento de um canal informal de comunicação consiste apenas na execução do método *DeleteResource* presente no *ResourcesServer*. Todos os demais tratamentos são realizados nos objetos *MemberParticipant* envolvidos.

O fechamento do canal informal é representado na figura 7.47.

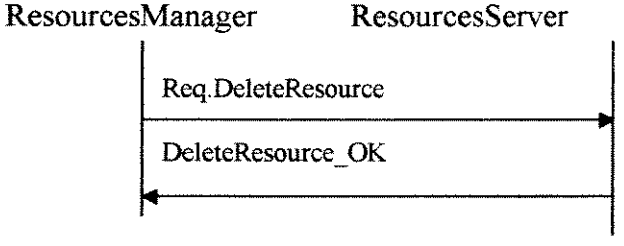


Figura 7.47 - Fechamento de um canal informal a nível do *ResourcesManager*

Protótipo:

CloseInformalChannel(member,IT_env)

Parâmetros:

- *member* (const char *): nome do próprio canal a ser fechado
- *IT_env*: estrutura para o tratamento de falhas de comunicação
- retorno do método (short): indicativo de erro de execução (0=ausência)

7.3.6.5 - *ResourcesManager::ShowFormalMessage*

A exibição de uma mensagem formal consiste, a nível do *ResourcesManager*, na evocação do método *ShowMessage* do objeto relativo ao recurso associado com o canal formal de comunicação (figura 7.48).

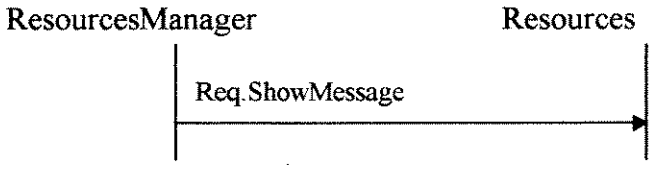


Figura 7.48 - Exibição de mensagem formal a nível do *ResourcesManager*

Protótipo:

ShowFormalMessage(message,smember,shost,IT_env)

Parâmetros:

- *message* (const char *): mensagem a ser exibida
- *smember* (const char *): proprietário da mensagem (membro corrente)
- *shost* (const char *): localização do membro corrente
- *IT_env*: estrutura para checagem de falhas de comunicação

7.3.6.6 - *ResourcesManager::ShowGeneralMessage*

Assim como a exibição de uma mensagem formal, o tratamento para a exibição de uma mensagem também consiste na evocação do método *ShowMessage* do objeto relativo ao recurso associado com o canal de mensagens gerais (figura 7.49).

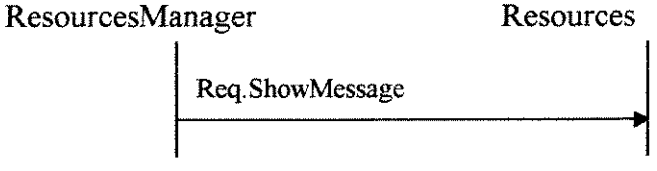


Figura 7.49 - Exibição de mensagem geral a nível do *ResourcesManager*

Protótipo:

ShowGeneralMessage(message,IT_env)

Parâmetros:

- *message* (const char *): mensagem a ser exibida
- *IT_env*: estrutura para checagem de falhas de comunicação

7.3.6.7 - *ResourcesManagerLocalResources::SetLCMemPar / ResourcesManagerLocalResources::SetLNMemPar / ResourcesManagerSharedResources::SetSCMemPar / ResourcesManagerSharedResources::SetSNMemPar*

Para o início das atividades cooperativas, há a necessidade da instanciação das estruturas responsáveis pelo armazenamento das referências dos membros associados aos gerenciadores de recursos (*MemPar*).

A distinção dos métodos acima listados consiste no tipo de membro (*chairman* ou *normal*) e a disposição dos recursos (*local* ou *compartilhado*). Sendo assim:

- *SetLCMemPar*: instancia um membro do tipo *chairman* em um gerenciador com recursos locais
- *SetLNMemPar*: instancia um membro do tipo *normal* em um gerenciador com recursos locais
- *SetSCMemPar*: instancia um membro do tipo *chairman* em um gerenciador com recursos compartilhados, colocando o relativo ponteiro na posição 0 do vetor *MemPar*
- *SetSNMemPar:* instancia um membro do tipo *normal* em um gerenciador com recursos compartilhados

Protótipo:

SetLCMemPar(mempar,IT_env) SetLNMemPar(mempar,IT_env)
SetSCMemPar(mempar,IT_env) SetSNMemPar(mempar,IT_env)

Parâmetros:

- *mempar*: ponteiro para o membro a ser armazenado, podendo ser: Chairman-Member ou NormalMember)
- *IT_env*: estrutura para a verificação de falhas de comunicação

7.3.6.8 - *ResourcesManagerLocalResources::ParcialCloseResources*

O fechamento parcial implica na possibilidade de uma possível reabertura dos recursos listados na lista à eles pertinentes, significando que tal processo somente realizará o fechamento de todos os recursos abertos no momento da evocação do método.

A figura 7.50 mostra a etapa do fechamento parcial, onde nota-se que o mesmo somente engloba a chamada do método *Kill* pertence ao objeto *Resource*.

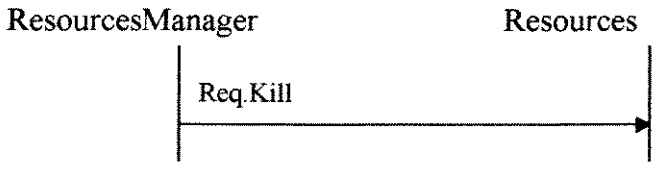


Figura 7.50 - Fechamento parcial dos recursos associados ao *ResourcesManager*

Protótipo:

ParcialCloseResources(IT_env)

Parâmetros:

- *IT_env*: estrutura para a verificação de falhas de comunicação
- retorno do método (short): indicativo de erro de execução (0=ausência)

7.3.6.9 - *ResourcesManagerLocalResources::CloseLGroup* *ResourcesManagerSharedResources::CloseSGroup*

Para a concretização do fechamento do grupo, onde todas as medidas a nível de políticas foram tomadas nas camadas superiores da plataforma *Groupware*, ocorre o fechamento dos recursos mediante sua destruição e a liberação da interface correspondente.

A figura 7.51 mostra o processo, onde nota-se a participação do objeto *ResourcesServer* e do *daemon* da plataforma de objetos distribuídos.

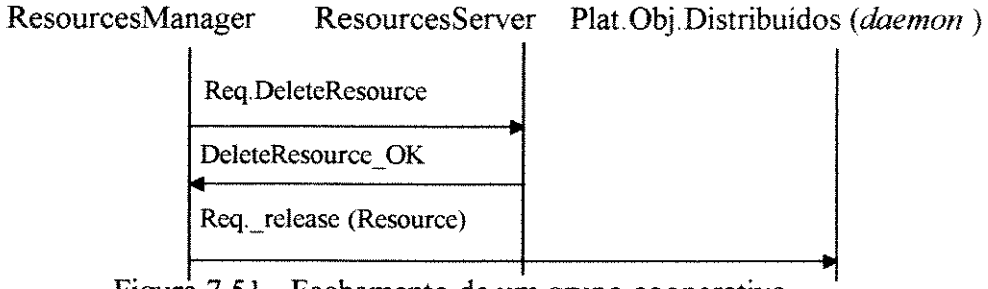


Figura 7.51 - Fechamento de um grupo cooperativo

Protótipo:

CloseLGroup(IT_env)
CloseSGroup(IT_env)

Parâmetros:

- *IT_env*: estrutura para verificação de falhas de comunicação
- retorno do método (short): indicativo de erro de execução (0=ausência)

7.3.6.10 - *ResourcesManagerSharedResources::ConnectMember*

O processo de conexão de membros corresponde apenas à sua introdução, não se manifestando as etapas de abertura dos recursos e instanciação dos ponteiros associados. Assim sendo, o método tem como função a instanciação, mediante a introdução no vetor *MembersProprietaries*, dos atributos do membro recém adicionado.

O método *SetInitialProprietary* desempenha o mesmo papel, porém adiciona o membro da posição inicial do vetor, o qual corresponde ao *chairman* do grupo cooperativo.

Protótipo:

ConnectMember(member,IT_env)

Parâmetros:

- *member* (const T_Member&): caracterísitcas associadas ao membro a ser introduzido
- *IT_env*: estrutura para a verificação de falhas de comunicação
- retorno do método (short): indicativo de erro de execução (0=ausência)

7.3.6.11 - *ResourcesManagerLocalResources::SetProprietary*

Para os grupos com recursos locais, há somente um membro associado com cada objeto *ResourcesManager*, não aparecendo o vetor *MembersProprietaries* mas apenas uma estrutura denominada *MemberProprietary*. A sua instanciação ocorre por intermédio da evocação do método *SetProprietary* por parte do *MembersManager*.

Protótipo:

SetProprietary(member,IT_env)

Parâmetros:

- *member* (const T_Member&): caracterísitcas assocoadas ao membro a ser introduzido
- *IT_env*: estrutura para a verificação de falhas de comunicação

7.3.7 - ResourcesObject

No protótipo alcançado da plataforma *Groupware* foi utilizado, para o gerenciamento das janelas, a plataforma *X-Window System* [Jones89].

7.3.7.1 - *TextualVideo::ShowMessage*

A nível dos objetos do tipo *Resources*, o processo de exibição de uma mensagem não faz distinção entre mensagens gerais, formais e informais, pois consiste apenas na execução dos métodos da camada gerenciadora de janelas (figura 7.52).

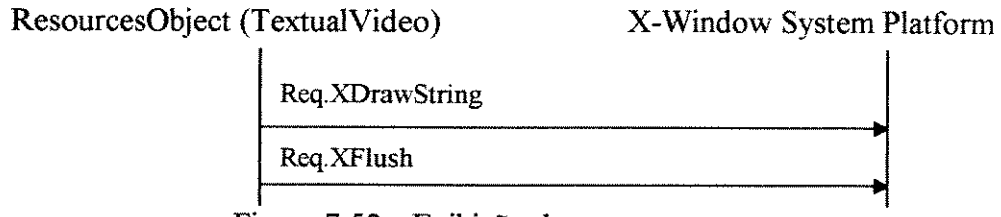


Figura 7.52 - Exibição de uma mensagem

A única diferenciação entre os tipos de mensagens encontra-se em seu conteúdo, confeccionado pelos objetos integrantes das camadas antecedentes.

Protótipo:

ShowMessage(sender,host,message,IT_env)

Parâmetros:

- *sender* (const char *): membro emissor da mensagem
- *host* (const char *): localização do emissor
- *message* (const char *): mensagem a ser exibida
- *IT_env*: estrutura para verificação de falhas

7.3.7.2 - *TextualVideo::OpenDisplay*

Para a abertura de um recurso do tipo *TextualVideo* faz-se uso dos métodos pertencentes à plataforma *X-Window System* conforme mostra a figura 7.53.

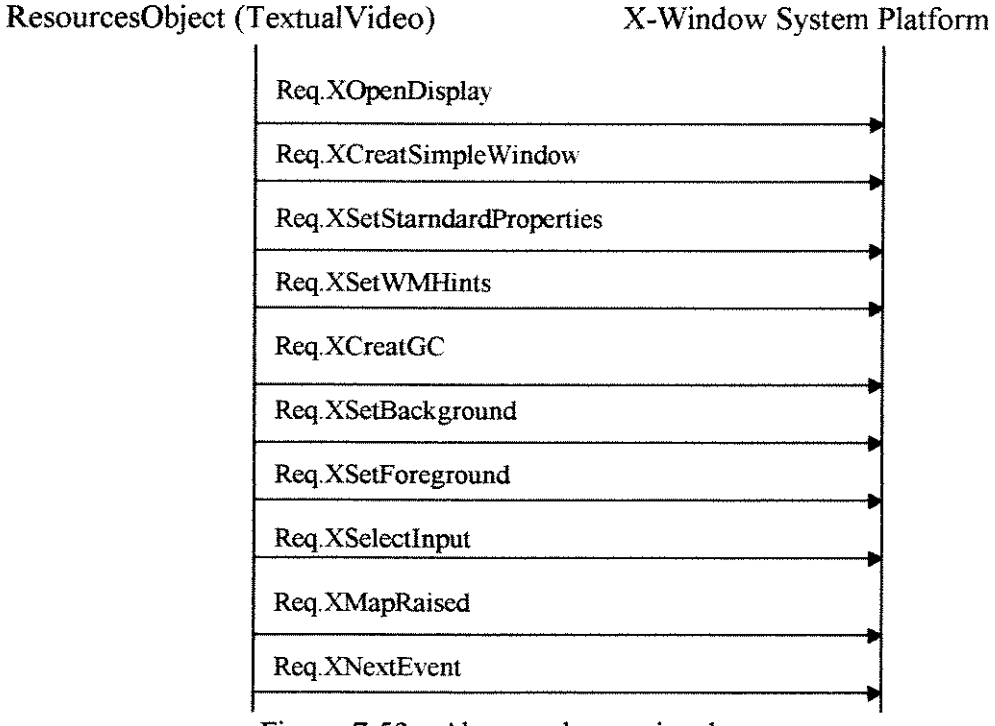


Figura 7.53 - Aberura de uma janela

Protótipo:

OpenDisplay(objname,err,IT_env)

Parâmetros:

- *objname* (const char *): utilizado para confecção do nome da janela
- *err* (short&): variável indicativa de erro
- *IT_env*: estrutura para checagem de falhas de comunicação

7.3.7.3 - *TextualVideo::Kill*

O processo de destruição de um recurso do tipo *TextualVideo* baseia-se no fechamento das estruturas que identificam o contexto gráfico, a janela e o "*display*" associados conforme ilustra a figura 7.54.

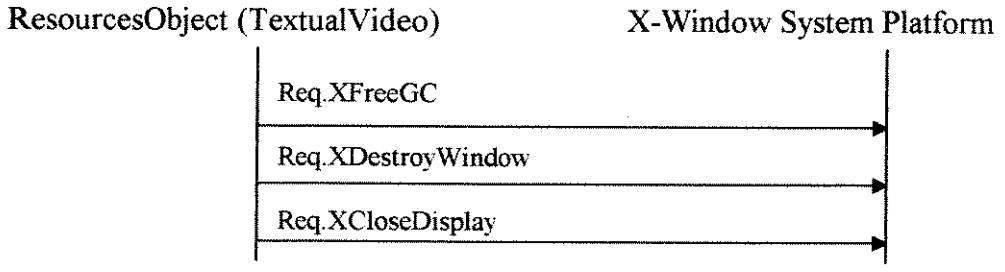


Figura 7.54 - Fechamento de uma janela

Protótipo:

Kill(IT_env)

Parâmetros:

- *IT_env*: estrutura para verificação de falhas de comunicação

7.4 - Objetos Servidores

A garantia da dinamicidade de criação e destruição de objetos é conquistada, também, por intermédio do surgimento de objetos servidores. Os objetos servidores, cuja instanciação é única em cada máquina abrangida pela plataforma, são responsáveis por criar e destruir objetos que integram a arquitetura da Camada Groupware.

Como exemplo, as figuras 7.55 a 7.58 ilustram um objeto do tipo *CSCWServices-Server*, recebendo as requisições do *ServiceFactory* para a criação e a destruição de objetos do tipo *CSCWServices*.

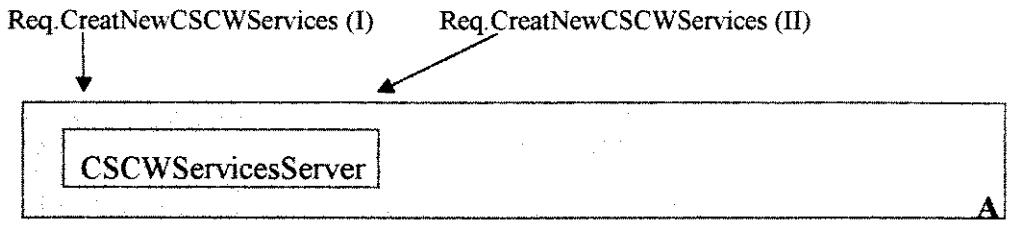


Figura 7.55 - Estado Inicial do Ambiente *A*

Na figura 7.55 é observado o estado inicial de um *host A*, havendo apenas um objeto servidor do tipo *CSCWServicesServer*. Os pedidos de criação provenientes do *ServiceFacto-ry* estão, neste caso, ainda trafegando pela rede, portanto, ainda não recebidos pelo servidor.

Após o recebimento das requisições, o *CSCWServicesServer* se incumbiu de criar os objetos *CSCWServices* solicitados conforme ilustra a figura 7.56. Nesta figura, suponha que está trafegando, em direção ao objeto servidor, uma mensagem proveniente do *ContactServices* requerendo o fechamento do grupo cooperativo (I).

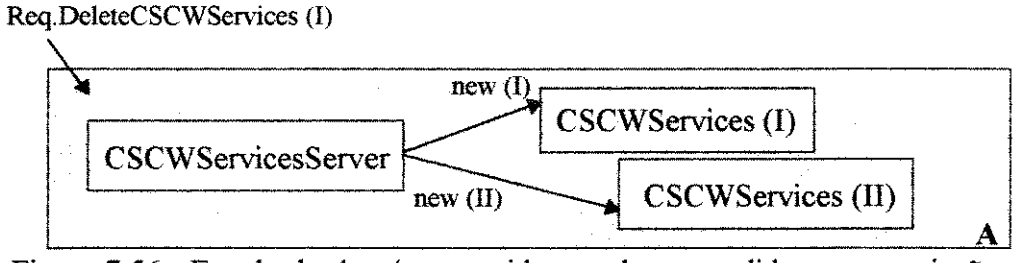


Figura 7.56 - Estado de *A* após o servidor receber os pedidos para a criação

Na figura 7.57, após recebida a mensagem para fechamento do grupo marcado como (I), o objeto servidor se encarrega do real fechamento.

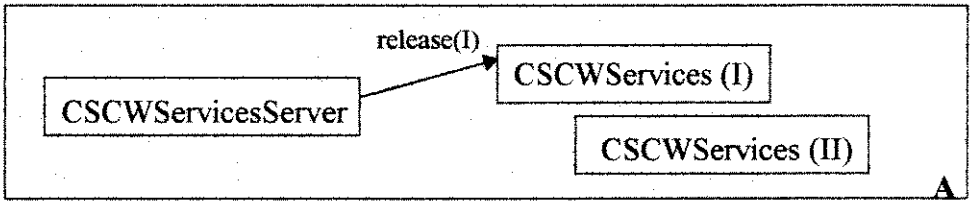


Figura 7.57 - Objeto Servidor atendendo ao pedido de fechamento do grupo

O estado final do ambiente *A*, após essa série de mensagens, é ilustrado no figura 7.58.

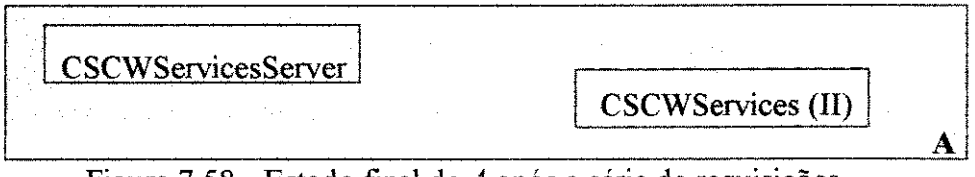


Figura 7.58 - Estado final de *A* após a série de requisições

Capítulo 8 - Exemplo de Utilização da Camada Groupware

8.1 - Introdução

Este capítulo aborda um exemplo de uma aplicação implementada sobre a Camada Groupware para a realização de uma tele-conferência conforme ilustra a figura 8.1.

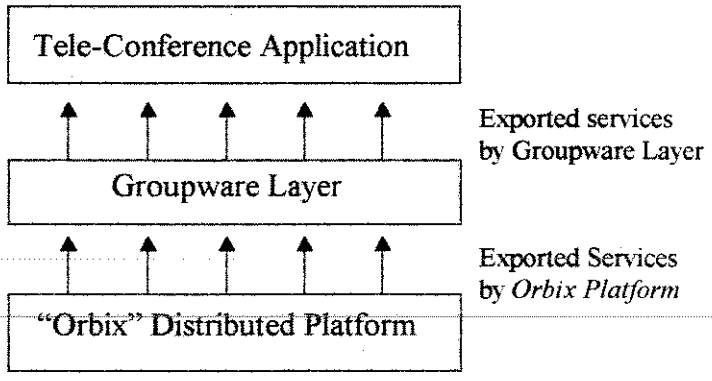


Figura 8.1 - Relacionamento entre a aplicação, a Camada Groupware e o Orbix

A construção de um ambiente cooperativo consiste na implementação de um aplicativo utilizando as funcionalidades da Camada *Groupware* representadas pelos serviços exportados pela mesma. Para tanto, há a necessidade de configuração do ambiente de distribuição, executando o núcleo de gerenciamento da plataforma de distribuição e armazenando as interfaces e implementações dos objetos da camada em seus respectivos repositórios.

A princípio, para a execução de um programa cooperativo, seguem-se três etapas:

- a própria implementação, culminando na "linkagem" com os módulos da plataforma.
- iniciação do *daemon* da plataforma de objetos distribuídos
- inserção das interfaces nos repositórios de interface

Para uma melhor exemplificação da *linkagem*, o anexo "B" mostra o arquivo *Makefile* utilizado para gerar o código do programa exemplo que é exposto no tópico seguinte, cuja listagem se encontra no anexo "C".

O anexo "D" trata de um arquivo de lote utilizado para adicionar, no repositório de interfaces, as informações relativas às interfaces dos objetos.

8.2 - Manipulação da Plataforma Distribuída *Orbix*

No *Orbix*, os componentes de um ambiente distribuído são representados pelos objetos e suas respectivas interfaces. Para tanto, as transações feitas entre as

partes cliente e servidor são realizadas por intermédio do *Daemon Orbix* e auxiliadas pelos repositórios de implementação.

O *daemon* é um aplicativo permanentemente em execução que recebe, dentre outras, as mensagens de requisição provenientes dos clientes e as mensagens de respostas vindas dos objetos servidores. A função do *daemon* é colocar ambas as partes em contato conforme é ilustrado na figura 8.2.

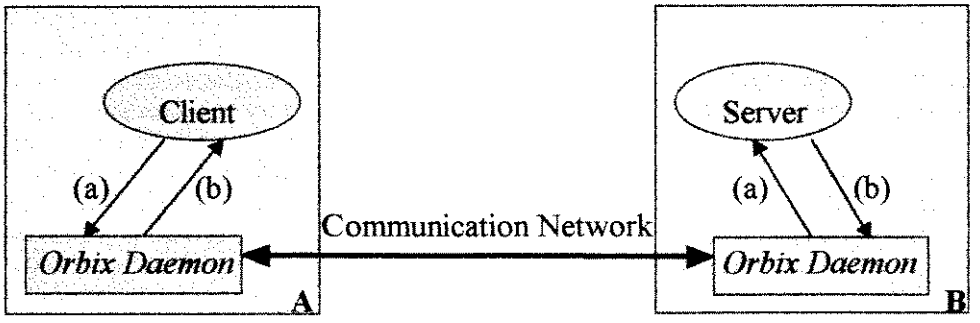


Figura 8.2 - Interação entre objetos cliente e servidor através do *daemon Orbix*

Na figura acima são exibidos dois *hosts* 'A' e 'B', os quais possuem, além do *daemon Orbix*, objetos cliente e servidor, respectivamente. Os *daemons*, por intermédio de uma rede de comunicação, envia os pedidos provenientes do objeto cliente (mensagens 'a') para o objeto servidor e devolverá o resultado do processamento para o cliente (mensagens denotadas por 'b').

No lado do servidor, surge a figura de uma importante estrutura: o repositório de implementação. Esta estrutura tem a função de mapeamento das interfaces dos objetos servidores instanciados sob o seu domínio, objetivando a correta entrega das mensagens.

Para a iniciação do *daemon* da plataforma *Orbix*, basta utilizar o programa "*orbixd*" (Orbix Daemon) e deixá-lo em execução no modo *background* e, para o registro das interfaces no repositório de interfaces, utiliza-se o comando "*putif*".

8.4 - Descrição da Aplicação de Tele-Conferência

Como mencionado anteriormente, o teste da Camada Groupware foi realizado segundo a implementação de um serviço cooperativo de tele-conferência, no qual os recursos são alocados individualmente para cada participante (recursos locais). Uma descrição geral deste exemplo encontra-se no tópico 6.5 (capítulo 6). A listagem do programa está inserida no anexo "C".

Quanto aos membros, estes podem ser classificados unicamente como *chairman* e membros normais, sendo que qualquer um pode assumir o direito de utilizar o canal formal de comunicação, assumindo o papel de detentor do direito de voz. A mudança do membro com direito de voz é realizada diretamente pelo *chairman*.

Caracteriza-se como *chairman* o membro que solicitou a abertura de um grupo cooperativo. Tal evocação consiste no envio de uma mensagem ao objeto *ContactService*, responsável pela instanciação do objeto auxiliar *ServiceFactory*, sendo que este possui a função de realmente iniciar e gerenciar as etapas de criação dos demais objetos integrantes da camada.

A interação entre os membros envolvidos na ação cooperativa consiste no emprego de janelas textuais, compreendendo três tipos:

- Janela para informações formais: Responsável pela transmissão das mensagens provenientes do membro detentor do direito de voz.

- Janela para informações gerais: Exibe as mensagens geradas pela manutenção da aplicação cooperativa, como por exemplo, a alteração do membro com direito de voz ou a entrada de algum novo membro ao grupo cooperativo.

- Janela para informações informais: Tem como objetivo a troca de informação informal entre dois membros.

Um membro terá em sua estação, no mínimo duas janelas: uma para as mensagens formais e outra para as gerais. Qualquer membro tem condições de abrir tantas janelas informais quanto forem necessárias.

A entrada dinâmica de novos membros após o início da ação cooperativa é realizada pelo serviço de questionamento ao *chairman*, cabendo à ele aceitar ou rejeitar a anexação do possível novo membro ao grupo.

Nota-se a existência de dois níveis de manipulação: o primeiro destina-se ao contato inicial com a camada cooperativa, caracterizando-se por intermédio de um menu que engloba chamadas aos serviços de contactação, e o segundo nível possibilita, a um membro já integrado nas ações cooperativas, executar os serviços de manutenção.

Fazem parte do primeiro menu as operações:

- Informações gerais sobre os grupos cooperativos já criados.
- Construir um novo grupo cooperativo.
- Iniciar as ações cooperativas de um grupo previamente criado.
- Iniciar ou restabelecer as ações cooperativas de um membro.
- Finalizar a ação cooperativa local.
- Fechar grupo cooperativo.
- Anexar-se a um grupo que já se encontra em andamento.

Por sua vez, o menu que compreende as operações da aplicação de tele-conferência compreende:

- Abrir canal informal de comunicação.
- Fechar canal informal de comunicação.
- Transmitir mensagem informal
- Transmitir mensagem formal.
- Mudar membro possuidor do direito de voz.
- Responder a questionamento.

Para o exemplo, um grupo cooperativo foi criado com a participação de dois membros localizados em máquinas diferentes de um mesmo domínio *internet*,

porém não há restrição em relação à localização das pessoas envolvidas pela ação cooperativa.

Em relação às políticas, o exemplo assume as seguintes características:

- participação dos membros: por imposição
- início da tele-conferência: por pedido do próprio chairman
- finalização do serviço: a pedido do chairman, sem votação entre os membros (imposta)
- interrupções do serviço: não permitidas
- ocorrência da finalização: imediatamente após o pedido
- qualidade dos membros: insubstituíveis, permitindo o desligamento temporário
- entrada de novos membros: pedidos ao chairman
- qualidade dos recursos: não poderão ser fechados, a não ser os canais informais de comunicação
- tempo máximo de utilização do canal formal: substituição do membro corrente dada pelo *chairman*
- utilização do objeto supervisor de QoS (*ViewSharer*): não

8.5 - Exemplificação dos Serviços Fornecidos pela Camada Groupware

Os ítens seguintes detalham a seqüenciação de evocações para executar as etapas de contactação e de manutenção do serviço de tele-conferência implementado. Em todos os métodos da camada *Groupware*, aparece a estrutura para a checagem de falhas de comunicação, denotada por *IT_X*, como último argumento.

Os ítens numerados de 8.4.1 a 8.4.7 necessitam de uma etapa de conexão com o objeto de contato da camada cooperativa. Tal conexão pode ser conseguida por intermédio do método *_bind*, conforme é codificado abaixo:

```
contac1=ContactService::_bind(":ContactServiceServer",
                              "aracaji.dca.fee.unicamp.br",IT_X);
```

Na linha acima, a variável *contac1*, do tipo *ContactService*, recebe a referência do objeto contido no servidor *ContactServiceServer* localizado na máquina *aracaji* dentro do domínio *dca.fee.unicamp.br*.

8.4.1 - Informações Gerais - *GeneralInfo()*

Após estabelecida a conexão com o objeto *ContactService*, informações sobre os grupos estabelecidos podem ser adquiridas mediante o método *GetCreatedInfo*. A linha abaixo mostra o caso de obtenção de tais informações:

```
contac->GetCreatedInfo(info,n,IT_X);
```

Na linha acima, as informações são armazenadas na variável *info* e o número de grupos cooperativos criados em *n*. Apesar do parâmetro *info* ser do tipo _IDL_SEQUENCE_T_Member, os campos *MemberName*, *MemberHost* e

MemberRole são utilizados para armazenar o nome dos grupos, suas localizações e os *chairmans* associados.

8.4.2 - Criação de Grupo Cooperativo - *MakeNewGroup()*

O segundo passo consiste na etapa de *bind* para o *ContactService* para que possa ser realizado o terceiro passo, representado pela execução do método *ContructNewService*.

```
ret=contac->ConstructNewService(cscwtype,servname,comment,start,duration,
                                memmantype, members,nummembers,
                                resmanID, policies,1, IT_X);
```

A linha extraída do exemplo enfoca o momento da chamada ao método de construção, responsável por disparar todas as etapas internas de confecção do grupo. O referido método retorna valor do erro, caso ocorrido, ou zero na ausência do mesmo.

8.4.3 - Iniciação do Cooperativismo Global - *InitGroup()*

O programa exemplo para tele-conferência cobre o último caso, ou seja, há a intervenção direta do *chairman* para o início das ações cooperativas, conforme a linha abaixo:

```
ret=contac->ActiveGroup(pos,IT_X);
```

A variável *pos* representa a posição ocupada pelo referido grupo no vetor de grupos criados, interno ao objeto *ContactService*. Tal posição é coletada por intermédio do método *SearchGroup*, referenciada na função *ConnectContactService* presente no mesmo arquivo exemplo.

8.4.4 - Iniciação do Cooperativismo Local - *InitMember()*

Assim como as ações globais, as locais passam também por uma etapa de iniciação. No caso das locais, o membro relacionado no grupo deve executar, após a conexão ao *ContactService*, as evocações dos métodos: *InitMember*, *SetIMemMan* e *SendGeneralMessage*.

A seqüência abaixo, extraída do código-fonte exemplo, ilustra as operações descritas.

```
ret=contac->InitMember(pos,LoginName,IT_X);
...
member=ChairmanMember::_bind(namebind,Host,IT_X);
...
member->SetIMemMan(IT_X);
...
member->SendGeneralMessage(message,IT_X);
...
```

A última evocação representa apenas um aviso aos demais membros sobre o início das atividades de um certo membro, portanto, é opcional em relação às etapas realmente necessárias para a iniciação do cooperativismo local.

8.4.5 - Fechamento do Grupo Cooperativo - *CloseGroup()*

O fechamento, neste exemplo por imposição do *chairman*, é realizado simplesmente pela invocação do método *CloseGroup*, conforme a linha de programação colocada a seguir:

if((ret=cont->CloseGroup(GroupName,IT_X))) return ret;

A única informação relacionada com o grupo a ser fechado passada para a camada *Groupware* consiste no seu próprio nome (*GroupName*). O teste da ocorrência de erro pode ser feito por intermédio do retorno do método, representando um valor diferente de zero caso a ocorrência de erro.

8.4.6 - Adição de Novo Membro a um Grupo em Execução - *AddNewMember()*

Para adicionar-se um membro durante a evolução das atividades cooperativas de um grupo, a aplicação deve permitir a entrada de todas as informações relativas ao mesmo, assim como as dos recursos que estarão à ele associados.

Após a introdução dessas informações e a conexão com o objeto *ContactService*, o usuário tem a possibilidade de evocar o método *AddNewMember*, conforme a linha abaixo:

if((ret=contac->AddNewMember(member,GroupName,resmanID,i,IT_X)))
return ret;

Os parâmetros do método acima utilizado, representam as informações do membro (*member*), nome do grupo no qual se deseja a incorporação (*GroupName*), descrição do gerenciador de recursos e dos recursos a serem alocados (*resmanID*) e a posição do grupo dentro da lista de grupos criados, interna ao *ContactService*. Esta posição é obtida pelo método *SearchGroup*, referenciada na função *ConnectContactService*.

8.4.7 - Interrupção das Ações Cooperativas Locais - *InterruptMember()*

Um membro pode se desligar temporariamente do grupo no qual está engajado, utilizando o método *Interrupt*, conforme a exemplificação abaixo:

if((ret=memb->Interrupt(LoginName,Host,IT_X))) return ret;

Para tanto, o objeto membro (*memb*) deve ser previamente conectado por intermédio do *bind*, onde a construção da identificação para o mesmo segue a seqüência:

```
strcpy(namebind,LoginName);
strcat(namebind,"_");
strcat(namebind,GroupName);
strcat(namebind,":MemberParticipantServer");
```

8.4.8 - Abertura de Canal Informal de Comunicação - *OpenInformalChannel(...)*

A abertura de um canal informal de comunicação consiste puramente na execução do método *OpenSInformalChannel* por parte do membro interessado em comunicar-se informalmente com algum outro pertencente ao mesmo domínio cooperativo.

```
ret=((ChairmanMember *)smember)->OpenSInformalChannel
        (TEXTUAL_VIDEO, dmember, LoginName, Host,IT_X);
```

A linha extraída do exemplo de tele-conferência ilustra a utilização do método para a abertura de canais de comunicação informal. Nela, observa-se como parâmetros: o tipo de recurso associado ao canal que, no caso acima, é indicado pela constante *TEXTUAL_VIDEO*; nome do membro destinatário (*dmember*); nome do membro requerente da abertura (*LoginName*); localização do membro solicitante (*Host*).

8.4.9 - Fechamento de Canal Informal de Comunicação - *CloseInformalChannel(...)*

O fechamento do canal informal de comunicação consiste na evocação do método *CloseInformalChannel* por parte do interessado no fechamento do mesmo, conforme ilustra a linha de código abaixo:

```
ret=((NormalMember*)smember)->CloseInformalChannel(dmember,
                                LoginName, Host,IT_X);
```

Para o exemplo acima, é suposta a existência um canal informal aberto entre os membros *smember* e *dmember*. Para o fechamento do canal, *smember* deve passar como argumentos o outro membro envolvido (*dmember*), além de seu próprio nome (*LoginName*) e a sua localização *(Host)*.

O método tem como retorno o valor do erro ocorrido ou zero caso o fechamento tenha sido realizado com sucesso.

8.4.10 - Transmissão de Mensagens Formais e Informais - *SendMessage(...)*

Os métodos para a transmissão de mensagens formais e informais são, no exemplo, polimórficos, porém, tratando-se dos serviços exportados pela camada, não o são. A manipulação dos dois tipos de mensagem diferenciam em relação ao membro destinatário: enquanto que a mensagem formal baseia-se no *broadcast* da mensagem a partir do membro corrente, a mensagem informal é representada pelo envio em *unicast*, relacionando apenas um membro receptador.

Para o envio de mensagens formais, evoca-se o método *SendFormalMessage*, conforme a linha abaixo:

*((NormalMember *)member)->SendFormalMessage(message,IT_X);*

Por outro lado, o envio de mensagens informais consiste na execução do método *SendInformalMessage*:

*((NormalMember *)member)->SendInformalMessage(message,dmember,IT_X);*

Os métodos ilustrados podem ser utilizados tanto por membros do tipo "*chairman*" quanto por aqueles pertencentes ao tipo "*normal*". Os conjunto de parâmetros é formado pela mensagem a ser enviada (*message*), identificação (nome) do membro destinatário (*dmember*) nas informais e pelo *IT_X*.

Um outro tipo de mensagem suportada pela camada cooperativa é coberta pelo canal de mensagens gerais, as quais representam mensagens destinadas às informações provenientes do próprio gerenciamento do serviço cooperativo ou da camada *Groupware*.

*((ChairmanMember *)member)->SendGeneralMessage(message,IT_X);*

A linha acima ilustra o modo de utilização do canal de mensagens gerais, onde o parâmetro *message* é enviado para todos os membros ativos e exibida na janela correspondente.

8.4.11 - Mudança Membro Corrente - *ChangeCurrentMember(...)*

Conforme a política adotada para a manipulação do canal formal de comunicação, a troca do membro que encontra correntemente utilizando-a, é realizada somente por imposição do *chairman*.

O exemplo codifica a ação de mudança utilizando dois serviços da camada *Groupware*: o primeiro para realizar a própria alteração do detentor de utilzação do canal formal e o segundo para avisar ao grupo o novo membro corrente. Tais serviços são denotados, respectivamente, por *ChangeCurrentMember* e *SendGeneralMessage*.

*ret=((ChairmanMember *)member)->ChangeCurrentMember(nm,IT_X);*

A linha acima pertence à codificação do programa exemplo, onde a variável "*nm*" representa o nome do novo membro que utilizará o canal formal de comunicação. O método tem como retorno um valor de erro, sendo zero a ausência deste.

8.4.12 - Responder Questionamento - *AnswerQuestion(...)*

No referido exemplo de tele-conferência, o único motivo para a utilização do serviço de questionamento é conseqüente da política adotada para a entrada de novos membros: vinculada à autorização do *chairman*, sendo este portanto, neste caso, o único membro autorizado a utilizá-lo.

Para realizar o envio da resposta basta evocar o método pertencente ao objeto *MemberParticipant* denotado por *AnswerQuestion* conforme a linha abaixo, extraída do exemplo.

```
if((ret=((ChairmanMember *)member)->AnswerQuestion(question,value,IT_X)))
    return ret;
```

Nota-se que a assinatura do método é constituída pela identificação da questão levantada (*question*), do respectivo valor da resposta (*value*) e da estrutura para checagem de erro de comunicação (*IT_X*). O método retorna o valor zero caso nenhum erro tenha ocorrido durante o o seu processamento interno e um valor diferente de zero para identificação de erros.

Capítulo 9 - Conclusões e Trabalhos Futuros

A crescente demanda de utilização de ambientes distribuídos fez surgir padrões que os caracterizassem, dentre eles destaca-se o Modelo de Referência ODP. A proposta contida no RM-ODP abrange importantes aspectos a serem seguidos na confecção de aplicações distribuídas.

O RM-ODP incorpora, além das características envolvidas com a distribuição de informações, recursos e de processamento, os seguintes conceitos:

- heterogeneidade
- mobilidade
- Qualidade de Serviço (QoS)
- autonomia dos domínios
- manipulação de federação

Além do referencial tecnológico, o Modelo de Referência ODP proporciona mecanismos que facilitam o processo de abstração e modelamento das aplicações cooperativas. Dentre os quais destaca-se a separação do problema nos pontos-de-vista empresarial, de informação, computacional, de engenharia e de tecnologia.

O suporte para as atividades relacionadas com os sistemas CSCW realça aspectos intimamente ligados com o ponto-de-vista de empresa e de informação. A modelagem de estruturas organizacionais e de atividades fornece um significante auxílio no contexto CSCW. Portanto, com a utilização dos pontos-de-vista de empresa e de informação, consegue-se abstrair as atividades, compartilhamentos e papéis desempenhados pelos membros dentro do grupo cooperativo.

A utilização destes resulta um conjunto de restrições e requisitos utilizado pelo ponto-de-vista computacional. Este visa estabelecer uma estruturação, em termos de objetos computacionais, das funcionalidades e características identificadas nos pontos-de-vista de empresa e de informação, permitindo identificar dois aspectos importantes do contexto CSCW: visibilidade e transparência.

Os dois últimos pontos-de-vista, constituídos pelo ponto-de-vista de engenharia e tecnológico, são utilizados para a própria implementação dos objetos CSCW, levando-se em conta os recursos disponíveis para tal.

Em relação à modelagem e implementação, o projeto da camada *Groupware* foi constituído de duas fases:

- delimitação da arquitetura da camada
- levantamento das interações entre os objetos integrantes da camada em relação à troca de mensagens.

Para a representação da arquitetura e interações entre os objetos, adotou-se o modelo OMT. A adoção deste modelo foi motivado por conseguir fornecer uma total e fiel abstração das funcionalidades necessárias para uma camada de suporte a

aplicações cooperativas dentro da filosofia de objetos distribuídos. Para tanto, utilizou-se a modelagem estática para o reconhecimento e disposição dos objetos pertencentes à camada e, a modelagem dinâmica, representada pelo histórico de eventos, para esquematizar o intercâmbio de mensagens entre os referidos objetos.

Sendo a Camada Groupware modelada segundo o paradigma de orientação a objetos, adotou-se a arquitetura CORBA devido ao seu eficiente mapeamento de objetos dentro da filosofia cliente/servidor.

O protótipo da camada *Groupware* foi implementado sobre o *Orbix* versão 1.0 como plataforma *CORBA* e sistema operacional AIX versão 3.2. Atualmente a camada de suporte a serviços cooperativos atende às exigências de uma ação cooperativa isócrona com janelas textuais como recursos, tal como por exemplo, uma aplicação de tele-conferência.

Encontrou-se algumas limitações do *Orbix*, como o não suporte às operações *multi-thread*, a não possibilidade de implementação de métodos polimórficos e a ausência de interfaces do tipo *stream*.

A Camada Groupware, assim como o exemplo de tele-conferência utilizado para teste, encontram-se disponíveis no Laboratório do Departamento de Computação e Automação Industrial.

Uma proposta para uma futura etapa, refere-se à incorporação das funcionalidades requisitadas pela manipulação de informações multimídia, assim como a anexação de uma base de dados orientada a objetos. Encontra-se também como planos futuros, a implementação dos serviços de suporte para as aplicações cooperativas que necessitem de uma constante verificação da qualidade de serviço.

Anexo A - Arquivos Pertencentes à Camada Groupware

Os arquivos de implementação da Camada *Groupware* podem ser agrupados, segundo sua funcionalidade em:

- Arquivos contendo os servidores de objetos
- Arquivos contendo os mecanismos de suporte às atividades cooperativas
- Arquivos contendo os mecanismos de comunicação entre os objetos da camada e a plataforma distribuída (*Orbix*)

Arquivos dos Servidores de Objetos

Abaixo são relacionados os arquivos que integram o escopo da classificação de objetos servidores (nome do arquivo-fonte → nome do objeto servidor associado).

- Contac_m.C → ContactServiceServer
- Factor_m.C → ServiceFactoryServer
- CSCWsv_m.C → CSCWServicesServer
- MemMan_m.C → MembersManagerServer
- MemPar_m.C → MemberParticipantServer
- ResMan_m.C → ResourcesManagerServer
- Resour_m.C → ResourcesServer

Arquivos para o Suporte às Atividades Cooperativas

As operações utilizadas pela camada para fornecer suporte às atividades cooperativas representam o conteúdo dos arquivos denotados por *objeto_i.o*. Integram os códigos construtores e destrutores, assim como todos os demais métodos dos objetos pertencentes à camada *Groupware*.

Para a definição das estruturas, das constantes e dos objetos, são auxiliados pelos arquivos *headers* denotados por *objeto_i.h*.

A seguir são listados todos os arquivos que compreendem essa categoria:

- Contac_i.C/Contac_i.h → Implementação do ContactService
- Factor_i.C/Factor_i.h → Implementação do ServiceFactory
- CSCWsv_i.C/CSCWsv_i.h → Implementação do CSCWServices
- MemMan_i.C/MemMan_i.h → Implementação do MembersManager
- MemPar_i.C/MemPar_i.h → Implementação do MemberParticipant
- ResMan_i.C/ResMan_i.h → Implementação do ResourcesManager
- Resour_i.C/Resour_i.h → Implementação do ResourcesObject

Suporte à Comunicação dos Objetos

Os métodos pertencentes aos módulos servidores e aos de suporte às atividades cooperativas necessitam de mecanismos de distribuição de mensagens para que possam realizar seus objetivos. O suporte à comunicação é conseguida por intermédio da plataforma de objetos distribuídos (*Orbix*) de acordo com a filosofia de cliente-servidor (figura A.1).

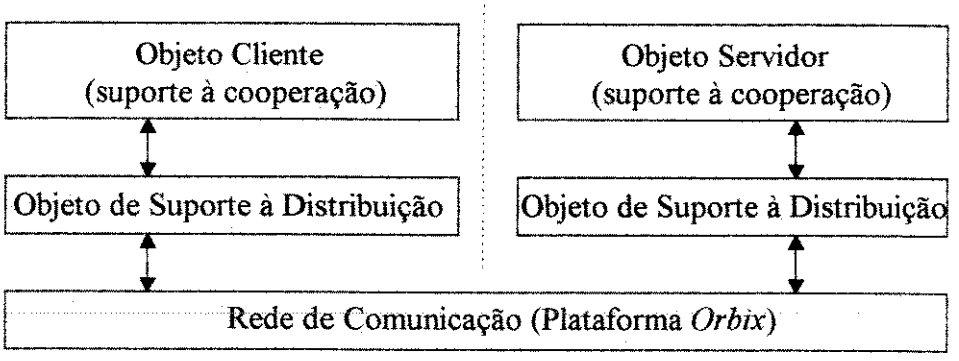


Figura A.1 - Representação da Interação entre as partes Cliente e Servidor

Para tanto, os mecanismos implementados para o suporte à comunicação estão inclusos nos arquivos denotados por *objetoC.C* e *objetoS.C*, referenciando aos métodos de objetos cliente e servidor, respectivamente. As definições para os referidos arquivos constam nos *headers* denotados por *objeto.hh*.

A relação a seguir exibe os arquivos de suporte comunicacional e os respectivos objetos abrangidos pelos mesmos:

- ContacC.C/ContacS.C/Contac.hh → ContactService
- FactorC.C/FactorS.CFactor.hh → ServiceFactory
- CSCWsvC.C/CSCWsvS.C/CSCWsv.hh → CSCWServices
- MemManC.C/MemManS.C/MemMan.hh → MembersManager
- MemParC.C/MemParS.C/MemPar.hh → MemberParticipant
- ResManC.C/ResManS.C/ResMan.hh → ResourcesManager
- ResourC.C/ResourS.C/Resour.hh → ResourcesObject
- GrpWreC.C/GrpWreS.C/GrpWre.hh → Definições gerais

Anexo B - Exemplo do Makefile

```
Contac_m.o: Contac_m.C Contac_i.h
        xlC -c Contac_m.C

Factor_m.o:Factor_m.C Factor_i.h
        xlC -c Factor_m.C

CSCWsv_m.o: CSCWsv_m.C CSCWsv_i.h
        xlC -c CSCWsv_m.C

MemMan_m.o: MemMan_m.C MemMan_i.h
        xlC -c MemMan_m.C

MemPar_m.o: MemPar_m.C MemPar_i.h
        xlC -c MemPar_m.C

ResMan_m.o: ResMan_m.C ResMan_i.h
        xlC -c ResMan_m.C

Resour_m.o: Resour_m.C Resour_i.h
        xlC -c Resour_m.C

Contac_i.o: Contac_i.C Contac_i.h
        xlC -c Contac_i.C

Factor_i.o: Factor_i.C Factor_i.h
        xlC -c Factor_i.C

CSCWsv_i.o: CSCWsv_i.C CSCWsv_i.h
        xlC -c CSCWsv_i.C

MemMan_i.o: MemMan_i.C MemMan_i.h
        xlC -c MemMan_i.C

MemPar_i.o: MemPar_i.C MemPar_i.h
        xlC -c MemPar_i.C

ResMan_i.o: ResMan_i.C ResMan_i.h
        xlC -c ResMan_i.C

Resour_i.o: Resour_i.C Resour_i.h
        xlC -c Resour_i.C


GrpWreC.o: GrpWreC.C GrpWre.hh
        xlC -c GrpWreC.C

ContacC.o:ContacC.C Contac.hh
        xlC -c ContacC.C
```

```
FactorC.o: FactorC.C Factor.hh
	xlC -c FactorC.C

CSCWsvC.o: CSCWsvC.C CSCWsv.hh
	xlC -c CSCWsvC.C

MemManC.o: MemManC.C MemMan.hh
	xlC -c MemManC.C

MemParC.o: MemParC.C MemPar.hh
	xlC -c MemParC.C

ResManC.o: ResManC.C ResMan.hh
	xlC -c ResManC.C

ResourC.o: ResourC.C Resour.hh
	xlC -c  ResourC.C

GrpWreS.o:GrpWreS.C GrpWre.hh
	xlC -c GrpWreS.C

ContacS.o:ContacS.C Contac.hh
	xlC -c ContacS.C

FactorS.o: FactorS.C Factor.hh
	xlC -c FactorS.C

CSCWsvS.o: CSCWsvS.C CSCWsv.hh
	xlC -c CSCWsvS.C

MemManS.o: MemManS.C MemMan.hh
	xlC -c MemManS.C

MemParS.o: MemParS.C MemPar.hh
	xlC -c MemParS.C

ResManS.o: ResManS.C ResMan.hh
	xlC -c ResManS.C

ResourS.o: ResourS.C Resour.hh
	xlC -c  ResourS.C

Contac: GrpWreS.o GrpWreC.o Contac_m.o ContacS.o Contac_i.o CSCWsvC.o  \
		SCWsv_i.o FactorC.o Factor_i.o
	xlC -o Contac GrpWreS.o GrpWreC.o Contac_m.o ContacS.o Contac_i.o \
		FactorC.o  CSCWsvC.o CSCWsv_i.o Factor_i.o \
		-L/proj/iona/Orbix1.3/lib -lITsrv -lITclt

Factor: GrpWreS.o GrpWreC.o Factor_m.o FactorS.o Factor_i.o CSCWsvC.o \
		CSCWsv_i.o ContacC.o
	xlC -o Factor GrpWreS.o GrpWreC.o Factor_m.o FactorS.o Factor_i.o \
		CSCWsvC.o ContacC.o CSCWsv_i.o -L/proj/iona/Orbix1.3/lib \
		-lITsrv -lITclt
```

```
CSCWsv: GrpWreS.o GrpWreC.o CSCWsv_m.o CSCWsvS.o CSCWsv_i.o \
        MemManC.o MemMan_i.o ContacC.o Contac_i.o
    xlC -o CSCWsv GrpWreS.o GrpWreC.o CSCWsv_m.o CSCWsvS.o \
        CSCWsv_i.o MemManC.o ContacC.o Contac_i.o MemMan_i.o \
        -L/proj/iona/Orbix1.3/lib -lITsrv -lITclt

MemMan: GrpWreS.o GrpWreC.o MemMan_m.o MemManS.o MemMan_i.o \
        MemParC.o MemPar_i.o CSCWsvC.o CSCWsv_i.o ContacC.o
    xlC -o MemMan GrpWreS.o GrpWreC.o MemMan_m.o MemManS.o  \
        CSCWsvC.o CSCWsv_i.o MemMan_i.o MemParC.o MemPar_i.o \
        ContacC.o -L/proj/iona/Orbix1.3/lib -lITsrv -lITclt

MemPar: GrpWreS.o GrpWreC.o MemPar_m.o MemParS.o MemPar_i.o \
        ResManC.o ResMan_i.o MemManC.o CSCWsvC.o ContacC.o \
        MemMan_i.o
    xlC -o MemPar GrpWreS.o GrpWreC.o MemPar_m.o MemParS.o \
        MemPar_i.o ResManC.o MemManC.o CSCWsvC.o  ContacC.o \
        MemMan_i.o ResMan_i.o -L/proj/iona/Orbix1.3/lib -lITsrv -lITclt

ResMan: GrpWreS.o GrpWreC.o ResMan_m.o ResManS.o ResMan_i.o ResourC.o \
        Resour_i.o MemParC.o MemManC.o CSCWsvC.o ContacC.o  \
        MemPar_i.o
    xlC -o ResMan GrpWreS.o GrpWreC.o ResMan_m.o ResManS.o \
        ResMan_i.o MemParC.o MemManC.o CSCWsvC.o ContacC.o  \
        MemPar_i.o ResourC.o Resour_i.o -L/proj/iona/Orbix1.3/lib -lITsrv \
        -lITclt

Resources: GrpWreS.o Resour_m.o ResourS.o Resour_i.o
    xlC -o Resources GrpWreS.o Resour_m.o ResourS.o Resour_i.o \
        -L/proj/iona/Orbix1.3/lib -lITsrv -L/usr/lib -lX11



CoopApp: GrpWreC.o CoopApp.o Contac_i.o ContacC.o MemPar_i.o MemParC.o \
        MemManC.o CSCWsvC.o
    xlC -o CoopApp GrpWreC.o CoopApp.o Contac_i.o ContacC.o \
        MemPar_i.o MemParC.o MemManC.o  CSCWsvC.o  \
        -L/proj/iona/Orbix1.3/lib -lITclt

CoopApp.o: CoopApp.C
    xlC -c CoopApp.C
```

Anexo C - Exemplo de uma Aplicação Cooperativa: Tele-Conferência

```
/*[][][][][][][][][][][][][][][][][][][][][][][]
  [] Arquivo: CoopApp.h        []
  [][][][][][][][][][][][][][][][][][][][][][][]*/

#ifndef COOPAPP_H
#define COOPAPP_H

#define MAX_OPTIONS 8

typedef struct
{
 char Options[MAX_OPTIONS][80];
 short NumberOfOptions;
} T_Menu;

enum Options1{INFO=1,CREAT,INITG,INITM,ENDM,ENDG,ADD,EXIT1};
enum Options2{OPEN_INFL=1,CLOSE_INF,SEND_INFL,SEND_FRML,
             CHNG_CURM, ASNQT,EXIT2};

short MakeNewGroup();
short ConnectContactService(ContactService*& contac,char *chairman,char *host,
                            short& pos);
short InitGroup();
short InitMember1(ContactService *contac,short pos);
short InitMember();
short InterruptMember();
short OpenInformalChannel(const char *dmember,void *smember);
short CloseInformalChannel(const char *dmember,void *smember);
short SendMessage(const char *message,void *member);
short SendMessage(const char *message,const char *dmember,void *smember);
short ServiceLoop();
short GeneralInfo();
short ChangeCurrentMember(const char *nm,void *member);
short CloseGroup();
short AddNewMember();
short AnswerQuestion(const char *question,short value,void *member);

#endif
```

```
/*[][][][][][][][][][][][][][][][][][][][][][][][][][][][]
 [] arquivo: CoopApp.c                    []
 [][][][][][][][][][][][][][][][][][][][][][][][][][][][]*/

#include "GrpWre.hh"
#include "Contac.hh"
#include "MemPar.hh"
#include "CoopApp.h"
#include <iostream.h>
#include <stdio.h>

T_Menu Menu1={{"[1] Informarcoes Gerais (todos)",
               "[2] Configurar Novo Grupo (todos)",
               "[3] Iniciar Cooperativismo (chairman)",
               "[4] Iniciar/Voltar Acao Local (membros)",
               "[5] Finalizar Acao Local (todos)",
               "[6] Finalizar Cooperativismo (chairman)",
               "[7] Incorporar-se a grupo (todos)",
               "[8] Finalizar CoopApp (todos)"},8};

T_Menu Menu2={{"[1] Abrir Canal Informal (todos)",
               "[2] Fechar Canal Informal (todos)",
               "[3] Transmitir Mensagem Informal (todos)",
              "[4] Transmitir Mensagem Formal (membro corrente)",
               "[5] Mudar Membro Corrente (chairman)",
               "[6] Responder Questionamento (chairman)",
               "[7] Voltar Menu Anterior"},7};

short SetChairman=0;
short InitiateMember=0;
char LoginName[25],Host[80],GroupName[80]="";

short MakeNewGroup()
{
 char servname[80],comment[255];
 short cscwtype,memmantype,nummembers,ret,i,j,npol;
 T_DataTime start;
 T_Time duration;
 ContactService *contac;
 _IDL_SEQUENCE_T_Member members(MAX_MEMBERS);
 _IDL_SEQUENCE_T_ResourcesManager resmanID(MAX_MEMBERS);
 _IDL_SEQUENCE_T_Policy policies(MAX_POLICIES);
 members._length=2;
 resmanID._length=MAX_MEMBERS;
 policies._length=MAX_POLICIES;
 for(i=0;i<MAX_MEMBERS;i++)
   {
    members._buffer[i].MemberName=new char[80];
    members._buffer[i].MemberHost=new char[80];
    members._buffer[i].MemberRole=new char[20];
    resmanID._buffer[i].Resources._buffer=new T_Resources[MAX_RESOURCES];
```

```
    resmanID._buffer[i].Resources._maximum=MAX_RESOURCES;
    resmanID._buffer[i].Resources._length=MAX_RESOURCES;
    for(j=0;j<MAX_RESOURCES;j++)
        {
         resmanID[i].Resources._buffer[j].ResourceName=new char[80];
         resmanID[i].Resources._buffer[j].ResourceHost=new char[80];
        }  //final for(int j...
    resmanID._buffer[i].Resources._length=2;
  }  //final for(int i...
for(i=0;i<MAX_POLICIES;i++) policies._buffer[i].PolicyName=new char[160];
strcpy(policies[0].PolicyName,"ADD_EXTERNAL_MEMBER");
policies[0].Value=1;
npol=1;
policies._length=1;
members._length=2;
strcpy(servname,"TESTE1");
strcpy(comment,"Nenhum Comentario (por enquanto...!)");
cscwtype=ISOSERV;
start.Day=start.Month=start.DayTime.Hour=start.DayTime.Minutes=0;
start.Year=0;
duration.Hour=duration.Minutes=0;
memmantype=ISO;
strcpy(members[0].MemberName,"sica");
strcpy(members[0].MemberHost,"aracaji.dca.fee.unicamp.br");
strcpy(members[0].MemberRole,"CHAIRMAN");
members[0].MemberType=CHAIRMAN;
members[1].MemberType=NORMAL;
strcpy(members[1].MemberName,"MembB");
strcpy(members[1].MemberHost,"camboriu.dca.fee.unicamp.br");
strcpy(members[1].MemberRole,"NORMAL");
nummembers=2;
resmanID[0].ResManType=resmanID[1].ResManType=LOCAL;
strcpy(resmanID[0].Resources[0].ResourceName,"GeneralMessages");
strcpy(resmanID[0].Resources[0].ResourceHost,members[0].MemberHost);
resmanID[0].Resources[0].ResourceType=TEXTUAL_VIDEO;
resmanID[0].Resources[0].MinQos=10;
resmanID[0].Resources[0].Formal=GENMESS;
strcpy(resmanID[0].Resources[1].ResourceName,"ChannelFormal");
strcpy(resmanID[0].Resources[1].ResourceHost,members[0].MemberHost);
resmanID[0].Resources[1].ResourceType=TEXTUAL_VIDEO;
resmanID[0].Resources[1].MinQos=10;
resmanID[0].Resources[1].Formal=FORMAL;
resmanID[0].NumberOfResources=2;
resmanID[0].MinQoS=10;
strcpy(resmanID[1].Resources[0].ResourceName,"GeneralMessages");
strcpy(resmanID[1].Resources[0].ResourceHost,members[1].MemberHost);
resmanID[1].Resources[0].ResourceType=TEXTUAL_VIDEO;
resmanID[1].Resources[0].MinQos=10;
resmanID[1].Resources[0].Formal=GENMESS;
strcpy(resmanID[1].Resources[1].ResourceName,"ChannelFormal");
strcpy(resmanID[1].Resources[1].ResourceHost,members[1].MemberHost);
resmanID[1].Resources[1].ResourceType=TEXTUAL_VIDEO;
resmanID[1].Resources[1].MinQos=10;
```

```
 resmanID[1].Resources[1].Formal=FORMAL;
 resmanID[1].NumberOfResources=2;
 resmanID[1].MinQoS=10;
 TRY
  {
   contac=ContactService::_bind(":ContactServiceServer",
                                "aracaji.dca.fee.unicamp.br",IT_X);
   if(!contac) return (-1);
  }
 CATCHANY
  {
   return (-2);
  }
 ENDTRY
 TRY
  {
   ret=contac->ConstructNewService(cscwtype,servname,comment,start,duration,
        memmantype, members,nummembers, resmanID,policies,1, IT_X);
   if(ret) return ret;
  }
 CATCHANY
  {
   return (-3);
  }
 ENDTRY
 return 0;
} //final MakeNewGroup

short ConnectContactService(ContactService **contac,char **chairman,char **host,
                            short& pos)
{
 char *chairman1,*host1;
 ContactService *contac1=*contac;
 short ret;
 TRY
  {
     contac1=ContactService::_bind(":ContactServiceServer",
                                   "aracaji.dca.fee.unicamp.br",IT_X);
  }
 CATCHANY
  {
   return (-4);
  }
 ENDTRY
 fflush(stdin);
 cout<<"\n\nGrupo: ";
 cin>>GroupName;
 cout<<endl<<endl;
 TRY
  {
   ret=contac1->SearchGroup(GroupName,chairman1,host1,pos,IT_X);
  }
```

```
 CATCHANY
  {
   return -9;
  }
 ENDTRY
 *contac=contac1;
 *chairman=chairman1;
 *host=host1;
 return 0;
}  //final ConnectContactService

short InitGroup()
{
 short ret,pos;
 char *chairman,*host;
 ContactService *contac;
 ret=ConnectContactService(&contac,&chairman,&host,pos);
 if(ret<0) return ret;
 if(strcmp(Host,host)) ret=2;
 if(strcmp(LoginName,chairman)) ret=3;
 if(!ret)
  {
   TRY
    {
     ret=contac->ActiveGroup(pos,IT_X);
     if(ret) return ret;
    }
   CATCHANY
    {
     return (-6);
    }
   ENDTRY
  }  //final if(ret...
 if(ret==1) cout<<"\n\nGrupo nao encontrado\n\n\n";
 if(ret==2) cout<<"\n\nGrupo encontrado em outro host\n\n\n";
 if(ret==3) cout<<"\n\nGrupo pertence a outro chairman\n\n\n";
 if(ret) return (-10);
 SetChairman=1;
 return 0;
}  //final InitGroup

short InitMember1(ContactService *contac,short pos)
{
 short ret;
 char namebind[160],message[255]="New State: ";
 TRY
  {
   ret=contac->InitMember(pos,LoginName,IT_X);
   if(ret) return ret;
  }
```

```
CATCHANY
 {
  return (-7);
 }
ENDTRY
strcpy(namebind,LoginName);
strcat(namebind,"_");
strcat(namebind,GroupName);
strcat(namebind,":MemberParticipantServer");
strcat(message,LoginName);
strcat(message," [");
strcat(message,Host);
strcat(message,"] ");
strcat(message," ==> ACTIVE");
if(SetChairman)
 {
  ChairmanMember *member;
  TRY
   {
     member=ChairmanMember::_bind(namebind,Host,IT_X);
   }
  CATCHANY
     {
       return -99;
     }
  ENDTRY
  TRY
     {
      member->SetIMemMan(IT_X);
     }
  CATCHANY
     {
      return -98;
     }
  ENDTRY
  TRY
     {
      member->SendGeneralMessage(message,IT_X);
     }
  CATCHANY
     {
      return -97;
   }
  ENDTRY
 }
else
 {
  NormalMember *member;
  TRY
     {
      member=NormalMember::_bind(namebind,Host,IT_X);
     }
```

```
    CATCHANY
        {
         return -99;
        }
    ENDTRY
    TRY
        {
         member->SetIMemMan(IT_X);
        }
    CATCHANY
        {
         return -98;
        }
    ENDTRY
    TRY
        {
         member->SendGeneralMessage(message,IT_X);
        }
    CATCHANY
        {
         return -97;
        }
    ENDTRY
   } //final else...
 return 0;
} //final InitMember1

short InitMember()
{
 short ret,pos;
 char *chairman,*host;
 ContactService *contac;
 ret=ConnectContactService(&contac,&chairman,&host,pos);
 if(ret<0) return ret;
 ret=InitMember1(contac,pos);
 if(ret) return ret;
 InitiateMember=1;
 return 0;
} //final Initmember

short OpenInformalChannel(const char *dmember,void *smember)
{
 short ret;
 if(!InitiateMember) return -18;
 if(SetChairman)
   {
    TRY
        {
         ret=((ChairmanMember *)smember)->OpenSInformalChannel
              (TEXTUAL_VIDEO,dmember,LoginName,Host,IT_X);
         if(ret) return ret;
        }
```

```
    CATCHANY
        {
         return -19;
        }
    ENDTRY
  } //final if...
 else
  {
   TRY
        {
         ret=((NormalMember *)smember)->OpenSInformalChannel
                (TEXTUAL_VIDEO,dmember,LoginName,Host,IT_X);
         if(ret) return ret;
        }
    CATCHANY
        {
         return -20;
        }
    ENDTRY
  } //final else...
 return 0;
} //final OpenInformalChannel

short CloseInformalChannel(const char *dmember,void *smember)
{
 short ret;
 if(!InitiateMember) return -16;
 if(SetChairman)
  {
   TRY
        {
         ret=((ChairmanMember *)smember)->CloseInformalChannel(dmember,
                                                LoginName,Host,IT_X);
         if(ret) return ret;
        }
    CATCHANY
        {
         return -17;
        }
    ENDTRY
  } //final if...
 else
  {
   TRY
        {
          ret=((NormalMember *)smember)->CloseInformalChannel(dmember,
                                                LoginName,Host,IT_X);
          if(ret) return ret;
        }
```

```
    CATCHANY
        {
         return -17;
        }
    ENDTRY
   } //final else...
 return 0;
} //final CloseInformalChannel

short SendMessage(const char *message,void *member)
{
 short ret=0;
 if(!InitiateMember) return -12;
 if(SetChairman)
   {
    TRY
      {
        ((ChairmanMember *)member)->SendFormalMessage(message,IT_X);
        if(ret) return ret;
      }
    CATCHANY
        {
         return -13;
        }
    ENDTRY
   } //final if...
 else
   {
    TRY
        {
         ((NormalMember *)member)->SendFormalMessage(message,IT_X);
         if(ret) return ret;
        }
    CATCHANY
        {
         return -13;
        }
    ENDTRY
   } //final else...
 return 0;
} //final SendMessage

short SendMessage(const char *message,const char *dmember,void *smember)
{
 short ret=0;
 if(!InitiateMember) return -14;
 if(SetChairman)
   {
```

```
    TRY
        {
         ((ChairmanMember *)smember)->SendInformalMessage(message,dmember,
                                                          IT_X);
         if(ret) return ret;
        }
    CATCHANY
        {
         return -15;
        }
    ENDTRY
   } //final if...
 else
   {
    TRY
        {
          ((NormalMember *)smember)->SendInformalMessage(message,
                                    dmember,IT_X);
          if(ret) return ret;
        }
    CATCHANY
        {
         return -15;
        }
    ENDTRY
   } //final else...
 return 0;
} //final SendMessage

short GeneralInfo()
{
 _IDL_SEQUENCE_T_Member info(MAX_SERVICES);
 short n;
 ContactService *contac;
 TRY
  {
   contac=ContactService::_bind(":ContactServiceServer",
                                "aracaji.dca.fee.unicamp.br",IT_X);
  }
 CATCHANY
  {
   return -21;
  }
 ENDTRY
 TRY
  {
   contac->GetCreatedInfo(info,n,IT_X);
  }
 CATCHANY
  {
   return -22;
  }
 ENDTRY
```

```
 if(!n)  cout << "\n\nNenhum Grupo Criado.\n\n";
 else for(short i=0;i<n;i++)
      {
       cout << "Grupo: " << info[i].MemberName;
       cout << "\nHost: " << info[i].MemberHost;
       cout << "\nChairman: " << info[i].MemberRole << endl << endl;
      }  //final for...
 for(short i=0;i<n;i++)
   {
    if(info[i].MemberName) delete(info[i].MemberName);
    if(info[i].MemberHost) delete(info[i].MemberHost);
    if(info[i].MemberRole) delete(info[i].MemberRole);
   }  //final for..
 if(info._buffer) delete(info._buffer);
 return 0;
}  //final GeneralInfo

short ChangeCurrentMember(const char *nm,void *member)
{
 short ret;
 char message[255]="New Current Member (Formal Channel): ";
 if(!SetChairman) return -23;
 TRY
   {
    ret=((ChairmanMember *)member)->ChangeCurrentMember(nm,IT_X);
    if(ret) return ret;
   }
 CATCHANY
   {
    return -24;
   }
 ENDTRY
 strcat(message,nm);
 TRY
   {
    ((ChairmanMember *)member)->SendGeneralMessage(message,IT_X);
   }
 CATCHANY
   {
    return -96;
   }
 ENDTRY
 return 0;
}  //final ChangeCurrentMember

short InterruptMember()
{
 short ret;
 char namebind[160];
 strcpy(namebind,LoginName);
 strcat(namebind,"_");
 strcat(namebind,GroupName);
 strcat(namebind,":MemberParticipantServer");
```

```
if(SetChairman)
  {
   ChairmanMember *memb;
   TRY
       {
          memb=ChairmanMember::_bind(namebind,Host,IT_X);
       }
   CATCHANY
       {
        return -21;
       }
   ENDTRY
   TRY
       {
        ret=memb->Interrupt(LoginName,Host,IT_X);
        if(ret) return ret;
       }
   CATCHANY
       {
        return -22;
       }
   ENDTRY
  }  //final if...
else
  {
   NormalMember *memb;
   TRY
       {
        memb=NormalMember::_bind(namebind,Host,IT_X);
       }
   CATCHANY
       {
        return -21;
       }
   ENDTRY
   TRY
       {
        ret=memb->Interrupt(LoginName,Host,IT_X);
        if(ret) return ret;
       }
   CATCHANY
       {
        return -22;
       }
   ENDTRY
  }  //final else...
 InitiateMember=0;
 SetChairman=0;
 GroupName[0]=0;
 return 0;
}  //final InterruptMember
```

```
short CloseGroup()
{
 short ret;
 ContactService *cont;
 if(!SetChairman) return -25;
 if(!strcmp(GroupName,"")) return -26;
 TRY
  {
   cont=ContactService::_bind(":ContactServiceServer",
                              "aracaji.dca.fee.unicamp.br",IT_X);
  }
 CATCHANY
  {
   return -27;
  }
 ENDTRY
 TRY
  {
   if((ret=cont->CloseGroup(GroupName,IT_X))) return ret;
  }
 CATCHANY
  {
   return -28;
  }
 ENDTRY
 TRY
  {
   cont->_release(IT_X);
  }
 CATCHANY
  {
   return -29;
  }
 ENDTRY
 InitiateMember=SetChairman=0;
 strcpy(GroupName,"");
 return 0;
}  //final CloseGroup

short AddNewMember()
{
 T_ResourcesManager resmanID;
 T_Member member;
 ContactService *contac;
 char *host,*chairman;
 short ret,i=0;
 member.MemberName=new char[80];
 member.MemberHost=new char[80];
 member.MemberRole=new char[80];
 resmanID.Resources._buffer=new T_Resources[MAX_RESOURCES];
```

```
 for(i=0;i<MAX_RESOURCES;i++)
  {
   resmanID.Resources._buffer[i].ResourceName=new char[80];
   resmanID.Resources._buffer[i].ResourceHost=new char[80];
  }  //final for...
 resmanID.Resources._maximum=MAX_RESOURCES;
 resmanID.Resources._length=3;
 strcpy(member.MemberName,"MembC");
 strcpy(member.MemberHost,"joaquina.dca.fee.unicamp.br");
 strcpy(member.MemberRole,"NORMAL");
 member.MemberType=NORMAL;
 member.QoS=10;
 resmanID.ResManType=LOCAL;
 strcpy(resmanID.Resources[0].ResourceName,"GeneralMessages");
 strcpy(resmanID.Resources[0].ResourceHost,member.MemberHost);
 resmanID.Resources[0].ResourceType=TEXTUAL_VIDEO;
 resmanID.Resources[0].MinQos=10;
 resmanID.Resources[0].Formal=GENMESS;
 strcpy(resmanID.Resources[1].ResourceName,"ChannelFormal");
 strcpy(resmanID.Resources[1].ResourceHost,member.MemberHost);
 resmanID.Resources[1].ResourceType=TEXTUAL_VIDEO;
 resmanID.Resources[1].MinQos=10;
 resmanID.Resources[1].Formal=FORMAL;
 resmanID.NumberOfResources=2;
 resmanID.MinQoS=10;
 if((ret=ConnectContactService(&contac,&chairman,&host,i))) return ret;
 TRY
  {
   if((ret=contac->AddNewMember(member,GroupName,resmanID,i,IT_X)))
        return ret;
  }
 CATCHANY
  {
   return -30;
  }
 ENDTRY
 return 0;
}  //final AddNewMember

short AnswerQuestion(const char *question,short value,void *member)
{
 short ret;
 if(!SetChairman) return -31;
 TRY
  {
   if((ret=((ChairmanMember *)member)->AnswerQuestion(question,value,IT_X)))
        return ret;
  }
 CATCHANY
  {
   return -32;
  }
 ENDTRY
```

```
 return 0;
}  //final AnswerQuestion

short ServiceLoop()
{
 void *member;
 char namebind[160];
 strcpy(namebind,LoginName);
 strcat(namebind,"_");
 strcat(namebind,GroupName);
 strcat(namebind,":MemberParticipantServer");
 if(SetChairman)
  {
   TRY
       {
        member=(ChairmanMember *)ChairmanMember::_bind(namebind,
                    Host,IT_X);
       }
   CATCHANY
       {
        return -16;
       }
   ENDTRY
  }  //final if...
 else
  {
   TRY
       {
        member=(NormalMember *)NormalMember::_bind(namebind,Host,IT_X);
       }
   CATCHANY
       {
        return -16;
       }
   ENDTRY
  }  //final else...
 short saida=0,op;
 for(;!saida;)
  {
   cout<"\n\n";
   for(short i=0;i<Menu2.NumberOfOptions;i++) cout<<Menu2.Options[i]<<endl;
   cout << "\n\n-->";
   cin>>op;
   switch(op)
       {
        case OPEN_INFL:
                    {
                     char dmember[80];
                     cout<<"\nMembro destino: ";
                     cin>>dmember;
                     cout<<"\n\n";
                     op=OpenInformalChannel(dmember,member);
                     if(op) cout<<"Erro na abertura: "<<op<<"\n\n";
```

```
                    else cout<<"Abertura com Sucesso.\n\n";
                    break;
                  } //final case OPEN_INFL
case CLOSE_INF:
                  {
                    char dmember[80];
                    cout<<"\nMembro destino: ";
                    cin>>dmember;
                    op=CloseInformalChannel(dmember,member);
                    if(op) cout<<"Erro no fechamento: "<<op<<endl;
                    break;
                  } //final case CLOSE_INF
case SEND_INFL:
                  {
                    char message[255],dmember[80];
                    cout<<"\nMembro destino: "; cin>>dmember;
                    cout<<"\nMensagem: ";      cin>>message;
                    cout<<endl;
                    op=SendMessage(message,dmember,member);
                    if(op) cout << "\nErro no envio: "<<op<<endl;
                    break;
                  } //final case SEND_INFL
case SEND_FRML:
                  {
                    char message[255];
                    cout<<"\nMensagem: ";cin>>message;
                    cout<<endl;
                    op=SendMessage(message,member);
                    if(op) cout << "\n\nErro no envio: "<<op<<endl;
                    break;
                  } //final case SEND_FRML
case CHNG_CURM:
                  {
                    char nmember[80];
                    cout << "\n\nNome: "; cin >> nmember;
                    cout << "\n\n";
                    op=ChangeCurrentMember(nmember,member);
                    if(op) cout<<"\n\nErro na mudanca: "<<op<<"\n\n";
                    else cout << "\n\nMudanca Realizada.\n\n";
                    break;
                  } //final case CHNG_CURM
case ASNQT:
                  {
                    char question[160];
                    short value;
                    cout << "\n\nQuestao: ";
                    cin >> question;
                    cout << "Valor (0=NAO;1=SIM): ";
                    cin >> value;
                    op=AnswerQuestion(question,value,member);
```

```
                        if(op) cout << "ERRO: " << op << "\n\n";
                        else cout << "Resposta enviada.\n\n";
                        break;
                      } //final case ASNQT...
        case EXIT2:
                     {
                       saida=1;
                       break;
                     } //final case EXIT2
      } //final switch
   op=0;
  } //final for...
 return 0;
} //final ServiceLoop

void main()
{
 short saida=0,op;
 cout<<"Login: "; cin>>LoginName;
 cout<<"Host: "; cin>>Host;
 cout << "\n\n\n";
 for(;!saida;)
  {
   for(short i=0;i<Menu1.NumberOfOptions;i++) cout<<Menu1.Options[i]<<endl;
   cout<<"\n\n--> ";
   cin>>op;
   switch(op)
       {
        case INFO:
            {
             GeneralInfo();
             break;
            } //final case INFO
        case CREAT:
            {
             op=MakeNewGroup();
             if(op) cout<<"\n\nERRO: "<<op<<"\n\n\n";
             else cout<<"\n\nCriacao com sucesso\n\n\n";
             break;
            } //final case CREAT
      case INITG:
            {
             op=InitGroup();
             if(!op) cout<<"\n\nGrupo agora corrente\n\n\n";
             else cout << "\n\nERRO: "<<op<<"\n\n\n";
             break;
            } //final case INITG
        case INITM:
            {
             op=0;
```

```
                if(!InitiateMember)
                  {
                   op=InitMember();
                   if(!op) cout << "\nMembro agora corrente\n\n\n";
                   else cout << "\n\nERRO: " << op << "\n\n\n";
                  } //final if...
                 if(op==315)
                  {
                   cout<<"Membro ja' se encontrava setado.Continuar?";
                   cout<<" <0> Sim    <1> Nao";
                   cin>>op;
                   cout<<"\n\n";
                  } //final if...
                 if(!op)
                  {
                   op=ServiceLoop();
                   if(op) cout << "\n\nERRO: " << op << "\n\n\n";
                   break;
                  } //final if...
              } //final case INITM
          case ENDM:
              {
                op=InterruptMember();
                if(!op) cout<<"\n\nSaida OK\n\n";
                else cout<<"\n\nERRO: "<<op<<"\n\n";
                break;
              } //final case ENDM
            case ENDG:
              {
                op=CloseGroup();
                if(op) cout<<"\n\nERRO: "<<op<<"\n\n";
                else cout<<"\n\nFechamento OK.\n\n";
                break;
              } //final case ENDG
            case ADD:
              {
                op=AddNewMember();
                if(op) cout << "\n\nERRO : "<<op<<endl<<endl;
                else cout <<"\n\nIncorporacao realizada.\n\n";
                break;
              } //final case ADD...
            case EXIT1: {saida=1;break;}
          } //final switch...
       op=0;
     } //final for...
  } //final main
```

Anexo D - Exemplo de Registro no Repositório de Interfaces

/proj/iona/Orbix1.3/bin/putit ContactServiceServer
/proj/mmedia/msc/sica/versao01.orb/Contac aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit ServiceFactoryServer
/proj/mmedia/msc/sica/versao01.orb/Factor aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit CSCWServicesServer
/proj/mmedia/msc/sica/versao01.orb/CSCWsv aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit MembersManagerServer
/proj/mmedia/msc/sica/versao01.orb/MemMan aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit MemberParticipantServer
/proj/mmedia/msc/sica/versao01.orb/MemPar aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit ResourcesManagerServer
/proj/mmedia/msc/sica/versao01.orb/ResMan aracaji.dca.fee.unicamp.br

/proj/iona/Orbix1.3/bin/putit ResourcesServer
/proj/mmedia/msc/sica/versao01.orb/Resources aracaji.dca.fee.unicamp.br

Referências Bibliográficas

[Anderson91] Anderson, D.P.; Homsy, G.; "*A Continuous Media I/O Server and Its Synchronization Mechanism*"; Computer IEEE, October 1991.

[Araújo96] Araújo, D.E.; "*Serviços de Gerenciamento ODP Utilizando a Arquitetura CORBA*"; DCA/FEEC/UNICAMP; Dissertação de Mestrado; Campinas, Setembro, 1996.

[Bannon91] Bannon, L.J.; Schmidt, K.; "*CSCW: Four Characters in Search of a Context*". Studies in Computer Supported Cooperative Work; J.M. Bowers and S.D. Benford (Editors); Elsevier Science Publishers B.V.; North-Holland, 1991.

[Bearman93] Bearman, M.Y.; "*ODP-Trader*"; Internacional Conference on Open Distributed Processing; Berlin, September, 1993.

[Blanning92] Blanning, R.W.; King, D.R.; Marsden, J.R.; Séror, A.C.; "*Intelligent Models of Human Organizations: The State of The Art*"; Journal of Organizational Computing, vol.2, no. 2, pp.123-129; 1992.

[Booch91] Booch, G.; "*Object Oriented Design with Applications*"; The Benjamin/Cummings Publishing Company Inc.; California, 1991

[Coad91] Coad, P.; Yourdon, E.; "*Object-Oriented Analysis*"; Yourdon Press Computing; Prentice-Hall Inc.; 2 ed.; New Jersey, 1991.

[Coulson92] Coulson, G.; Smalley, J.; Blair, G.S.; "*The Design and Implementation of a Group Invocation Facility in ANSA*"; Technical Report MPG-92-34; Department of Computing, Lancaster University; Lancaster, UK, 1992.

[Ellis91] Ellis, C.A.; Gibbs, S.J; Rein, G.L.; "*Groupware: Some Issues and Experiences*"; Communications of the ACM; vol. 34; no.1; January, 1991

[Gerson86] Gerson, E.M., Star, S.L.; "*Analyzing Due Process in the Workplace*"; ACM Transactions on Office Information Systems, vol. 4, no. 3, July 1986, pp. 257-270.

[Greif88] Greif, I. "*Computer-Supported Cooperative Work: A Book of Readdings*". Morgan Kaufmann, 1988.

[Griethuysen92] Griethuysen, J.J.; "*Enterprise modelling, a necessary basis for modern information systems*", Open Distributed Processing; J.de Meer, V. Heymer and R. Roth (Editors) ; Elsevier Science Publishers B.V.; North-Holland, 1992. IFIP

[Gunji95] - Gunji, T.; "*Plataforma Multiware: Serviços de Objetos*", DCA-FEEC-UNICAMP; Dissertação de Mestrado; Campinas, Agosto 1995.

[Herbert92] Herbert, A. "*The Challenge of ODP*"; Open Distributed Processing; J.de Meer, V. Heymer and R. Roth (Editors) ; Elsevier Science Publishers B.V.; North-Holland, 1992; IFIP.

[Jones89] Jones, O.; "*Introduction to the X-Window System - Updated for XVIIR4*"; Prentice-Hall Inc.; New Jersey, 1989.

[Lee93] Lee, K; et ali; "*A Framework for Controlling Cooperative Agents*"; IEEE Computer; July, 1993.

[Linington92] Linington, P.F.; "*Introduction to Open Distributed Processing Basic Reference Model*"; Open Distributed Processing; J.de Meer, V. Heymer and R. Roth (Editors) ; Elsevier Science Publishers B.V.; North-Holland, 1992. IFIP.

[Loyolla94a] Loyolla, W.P.D.C; Fontes, I. R.; Sica, F. C.;Mendes, M. J.; "*Plataforma Multiware: Especificação da Camada de Suporte para Aplicações CSCW*"; 12º SBRC - Sociedade Brasileira de Redes de Computadores, Curitiba, Maio de 1994.

[Loyolla94b] Loyolla, W.P.D.C.;Madeira, E.R.M; Mendes, M.J.; Cardozo, E.; Magalhães, M.F; "*Multiware Plataform: an Open Distributed Environment for Multimedia Cooperative Applications*"; IEEE Computer Software & Applications Conference - COMPSAC'94; Taipei, Taiwan, November 1994.

[Maezi95] - Maezi, M.; "*Plataforma Multiware: Interface de Programação*", DCA-FEEC-UNICAMP; Dissertação de Mestrado; Campinas, Agosto 1995.

[Mendes93] Mendes, M.J.; Loyolla, W.P.D.C.; Madeira, E.R.M.; "*Demos: A Distributed Decision Making Open Support System*"; 4th Workshop On Future Trends of Distributed Computing Systems do IEEE Computer Society, Lisbon September 1993.

[OMG92] Object Management Architecture Guide, version 2.0, OMG TC Document 92.11.1, September 1992.

[OMG93] Object Management Group; "*Object Request Broker Architecture*"; OMG TC Document 93.7.2; July, 1993.

[OMG96] Object Management Group; "*The Common Object Request Broker: Architectures and Specification Version 2.0*"; OMG TC Document 96-03-04; July 1995.

[Orbix95a] IONA Technologies Ltd; "*Orbix - Distributed Object Technology - Release 1.3.1*"; February, 1995.

[Orbix95b] IONA Technologies Ltd; "*Orbix - Distributed Object Technology - Advanced Programmer's Guide - Release 1.3.1*"; February, 1995.

[Navarro92] Navarro, L.; Prinz, W.; Rodden, T.; "*Towards Open CSCW Systems*"; IEEE; July 1992.

[Page-Jones88] Page-Jones, M.; "*Projeto Estruturado de Sistemas*"; Makron Books do Brasil Editora Ltda; São Paulo, 1988.

[Raymond93] Raymond, K.A.; "*Reference Model of Open Distributed Processing: a Tutorial*"; Proceedings of the IFIP TCG/WG6.1 Internacional Conference on Open Distributed Processing; Berlin, 1993.

[Ricarte96] Ricarte, I.L.M.; Tobar, C.M.; "*Towards an Architecture for Distributed Multimedia Databases*"; Internacional Conference on Intelligent Information Systems; Washington, DC; June 1996.

[RM-ODPa] - ISO - 10746-1 RM-ODP, "*Part 1: Overview and Guide To Use*", May 1995.

[RM-ODPb] - ISO - 10746-2 RM-ODP, "*Part 2: Descriptive Model*", June 1995.

[RM-ODPc] - ISO - 10746-3 RM-ODP, "*Part 3: Prescriptive Model*", June 1995.

[RM-ODPd] - ISO - 10746-4 RM-ODP, "*Part 4 - Architectural Semantics*", June 1995.

[Rodden92] Rodden, T.; Blair, G.S.; "*Distributed systems support for computer supported cooperative work*"; Computer Comunications, vol. 15, no.8, Octuber, 1992.

[Rosenberg92] Rosenberg, J; et ali; "*Multimedia Communications for Users*", IEEE Communications Magazine; May, 1992.

[Rudkin92] Rudkin, S.; "*Modelling Object Interfaces for Open Distributed Processing*"; Bt Technol. J.; Vol 10, number 2; April 1992.

[Rumbaugh91] Rumabugh, J.; et ali; "*Object-Oriented Modeling and Design*"; Prentice-Hall Inc.; New Jersey, 1991.

[Shlaer88] Shlaer, S.; Mellor, S.J.; "*Object-Oriented System Analysis - Modeling the World in Data*"; Prentice-Hall Inc.; New Jersey, 1988.

[Tanenbaum92] Tanenbaum, A.S.; "*Modern Operating Systems*"; Prentice-Hall Inc.; New Jersey, 1992.

[Takahashi90] T.Takahashi; Liesenberg, H.K.E; Xavier, D.T.; "*Programação Orientada a Objetos - Uma Visão do Paradigma de Objetos*". VII Escola de Computação. São Paulo, 1990.

[Tizzo95] Tizzo, N.P.; "*Sincronização de Dados Multimídia em Ambiente Aberto Distribuído*"; DCA-FEEC-UNICAMP; Dissertação de Mestrado; Campinas, 1995.

[Tobar94] Tobar, C.M.; Nevermann, H.;Ricarte, I.L.M.; "*Multiware: A Distributed Object Database System for Multimedia Support*"; Draft Version; Dept.of Computer Engineering and Industrial Automation; State University of Campinas. Campinas, September, 1994.

[Tobar95] Tobar, C.M.; Ricarte, I.L.M.; "*Multiware Databse: A Distributed Object Database System for Multimedia Support*"; In Open Distributed Processing: Experiences with Distributed Environments; Proceedings of the IFIP International Conference on Open Distributed Processing, ICODP'95, pp.439-450; February 1995.

[Tschammer93] Tschammer, V.; Mendes, M.J.; Souza, W.L.; Madeira, E.R.M.; Loyolla, W.P.D.C.; "*Processamento Distribuído Aberto e o Modelo RM-ODP/ISO*". Simpósio Brasileiro de Redes de Computadores; May, 1993.

[Williams92] Williams, N.; Blair, G.; Coulson, G.; Davies, N.; Rodden, T.; "*The Impact of Distributed Multimedia Systems on Computer Support for Cooperative Work*"; Technical Report MPG-92-43; Department of Computing, Lancaster University; Lancaster, UK, 1992.

[Wilson91] Wilson, P.; "*Computer Supported Cooperative Work - An Introduction*"; Intellect; Oxford, England, 1991.

[Winograd86] Winograd, T.; "*A language/action perspective on the design of cooperative work*"; CSCW'86 Proceedings. Conference on Computer-Supported Cooperative Work. Austin, Texas, ACM, December (3-5), 1986; pp. 203-220.

[Yamazaki93] Yamazaki, S.; et ali.; "*Object-Oriented Design of Telecommunication Software*"; IEEE Software; January, 1993.