ULISSES MANTOVAN

UMA ABORDAGEM, BASEADA EM FRAMEWORK E NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE, PARA O DESENVOLVIMENTO DE SISTEMAS DE ARQUIVOS PARALELOS DISTRIBUÍDOS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2006

ULISSES MANTOVAN

UMA ABORDAGEM, BASEADA EM FRAMEWORK E NA TÉCNICA DE DESCRIÇÃO FORMAL ESTELLE, PARA O DESENVOLVIMENTO DE SISTEMAS DE ARQUIVOS PARALELOS DISTRIBUÍDOS

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Wanderley Lopes de Souza

São Paulo
2006

FICHA CATALOGRÁFICA

Mantovan, Ulisses
Uma abordagem, baseada em framework e na Técnica de Descrição Formal Estelle, para o desenvolvimento de Sistemas de Arquivos Paralelos Distribuídos / U. Mantovan. -- São Paulo, 2006.
266 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Descrição de sistemas 2.Arquiteturas paralelas 3.Simulação distribuída 4.Frameworks I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Ao Prof. Dr. Wanderley Lopes de Souza, pela orientação segura que tornou possível a realização desse trabalho.

À Prof[a]. Dr[a]. Liria Matsumoto Sato, pela presteza e dedicação ao projeto.

Aos colegas do Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo pelo apoio e incentivo

A todas as pessoas que, de alguma forma, contribuíram para a obtenção deste importante objetivo na minha carreira acadêmica.

RESUMO

O constante aumento da velocidade de processamento, devido principalmente à utilização de um número cada vez maior de processadores, tem propiciado grandes avanços no projeto e na construção de sistemas computacionais paralelos. Entretanto o desempenho de muitas aplicações é afetado pela latência das operações de Entrada e Saída de dados. Para solucionar esse problema, sistemas de arquivos paralelos, que oferecem acesso paralelo aos dados armazenados em diversos discos, vêm sendo desenvolvidos. O desenvolvimento desses sistemas complexos pode ser beneficiado pela adoção de Técnicas de Descrição Formal (TDFs), durante as fases de projeto e especificação dos mesmos, as quais podem ser aliadas a técnicas de implementação durante as demais fases. Neste sentido, este projeto propõe uma abordagem baseada em *frameworks* e na TDF *Extended State Transition Language* (Estelle), para a especificação formal, validação, implementação e teste de sistemas dessa categoria. Um *framework* conceitual que descreve um sistema funcional é apresentado, e dois estudos de caso são desenvolvidos dando origem a dois sistemas de arquivos derivados do *framework*. Uma metodologia para a validação, que usa ferramentas de simulação, é apresentada. Um dos estudos de caso é implementado semi-automaticamente, a partir de sua especificação formal Estelle, e comparações de desempenho com o mesmo sistema implementado manualmente são realizadas.

ABSTRACT

The constant increase of processing speed, mainly due to the use of a large number of processors, has allowed an improvement in the design and building of parallel computation systems. However, the performance of several types of applications is affected by the latency originated from Input/Output operations on data. In order to solve this problem parallel file systems, which allow parallel access to the data stored on a set of discs, have been developed. The design of such complex systems can benefit from the adoption of implementation techniques allied with Formal Description Techniques (FDTs). Aimed to introduce the use of FDTs in the development cycle of distributed parallel file systems, this work proposes an approach, based on framework and the FDT Extended State Transition Language (Estelle), for the formal specification, validation, implementation and testing of systems belonging to this domain. A conceptual framework that describes a basic functional system is presented, and two case studies are developed from it. A methodology for Estelle specification validation that makes use of simulation tools is also proposed in this work. One of the systems, developed as a case study, is semi-automatically implemented from its Estelle formal specification, and performance comparisons with a hand-coded implementation of the same system are done.

SUMÁRIO

LISTA DE FIGURAS
LISTA DE TABELAS

1 Introdução 1
1.1 Justificativa 1
1.2 Sistemas de Arquivos Paralelos Distribuídos 3
1.2.1 Distribuição dos Dados 4
1.2.2 Organização de Arquivos Paralelos e Primitivas de Acesso 5
1.2.3 Arquitetura dos Sistemas de Arquivos Paralelos Distribuídos 7
1.3 Metodologia 12
1.4 Organização do Texto 14

2 Técnicas de Descrição Formal 15
2.1 LOTOS 16
2.1.1 Conceitos Básicos 16
2.1.2 LOTOS Básico 18
2.1.2.1 Processos e Operadores Básicos 18
2.1.2.2 Eventos Internos 19
2.1.2.3 Composição Paralela e Seqüencial 20
2.1.3 Tipos de Dados 21
2.1.4 LOTOS Completo 22
2.1.5 Especificação de Sistemas 24
2.2 E-LOTOS 24
2.2.1 A Linguagem Base 25
2.2.1.1 Declarações 25
2.2.1.2 Expressões de Comportamento 28
2.2.2 Linguagem Modular 34
2.2.2.1 Interfaces 35
2.2.2.2 Módulos 35
2.2.2.3 Módulos Genéricos 36
2.3 SDL 36
2.3.1 Modelagem de Sistemas em SDL 37
2.3.2 Comunicação 39
2.3.2.1 Comunicação por Sinais 39
2.3.2.2 Compartilhamento de Variáveis 41
2.3.3 Comportamento 42
2.4 Estelle 44
2.4.1 Modelagem de Sistemas em Estelle 45
2.4.2 Estrutura de Módulos 47
2.4.3 Comunicação 48
2.4.4 Cabeçalho de Módulo 49
2.4.5 Corpo de Módulo 50
2.4.6 Criação de Instâncias e de Links de Comunicação 54
2.4.7 Semântica 56
2.4.7.1 Situações Globais 57
2.4.7.2 Relação de Próxima Situação 58
2.5 Comparação entre as TDFs 59
2.6 Ferramentas para TDFs 61

3 Abordagem Baseada em um Framework Conceitual e em Estelle 68
3.1 Framework Conceitual para um Sistema Básico 69
3.1.1 Primitivas de Serviço 70
3.1.2 Arquitetura do Framework Conceitual 72
3.1.2.1 *Unified Modeling Language* (UML) 72
3.1.2.2 Framework Conceitual 74
3.2 Escolha da TDF para a descrição formal do Framework Conceitual 76
3.3 Especificação Formal do Sistema Básico em Estelle 77
3.3.1 Refinamento da Arquitetura 81
3.3.1.1 O Módulo Cliente 82
3.3.1.2 O Módulo Servidor 83
3.4 Complementação da Especificação 84
3.4.1 Mapeamento de MEFEs para construções Estelle 85
3.5 Validação da Especificação 88
3.5.1 Aspectos considerados durante a validação 89
3.5.2 Cenários de Simulação 90
3.6 Implementação da Especificação 91

4 Primeiro Estudo de Caso: Sistema Galley 93
4.1 Especificação e Validação do Sistema Galley 94
4.2 Protocolo de Comunicação 94
4.3 Primitivas de Serviço 96
4.4 Comportamento dos Processos 97
4.5 Mapeamento das MEFEs para construções Estelle 100
4.6 Definição das interações entre processos 102
4.7 Definição de funções que armazenam informações internas aos módulos 102
4.8 Localização dos módulos na Rede 103
4.9 Mecanismo de retransmissão 104
4.10 Estratégia de Distribuição de Dados 104
4.11 Validação do Sistema Descentralizado 105
4.11.1 Ambiente de Simulação 106
4.11.2 Cenários de Simulação 107
4.11.3 Testes das Operações de Leitura e Escrita 114

5 Segundo Estudo de Caso: Network Parallel File System (NPFS) 119
5.1 Protocolo de Comunicação 120
5.2 Primitivas de Serviço 122
5.3 MEFEs dos Processos 123
5.4 Especificação formal do Sistema Centralizado em Estelle 125
5.4.1 Refinamento da Especificação 128
5.4.2 O Módulo Mestre 129
5.5 Especificação formal segundo o framework conceitual 130
5.6 Simulação do Sistema Centralizado 131

6 Implementação e Teste de Sistemas 135
6.1 Implementação do sistema NPFS 136
6.1.1 Modificações na Arquitetura Inicial 136
6.1.2 O Módulo de Rede 139
6.1.3 Complementação da Especificação 140

6.1.4 Decisões de Implementação 141
6.2 Simulação com as Funções de Rede e das Funções Primitivas Implementadas 144
6.3 Considerações sobre a Simulação com as Funções de Rede e Funções Primitivas Implementadas 146
6.4 Implantação e Desempenho do Sistema 147
6.4.1 Entrada e Saída de dados 148
6.4.2 Análise da Implementação 150

7 Conclusões 152
7.1 Contribuições 154
7.2 Projetos Futuros 155

LISTA DE REFERÊNCIAS 156

Anexo A: Diagramas de Transição de Estados 162
A.1 Diagrama de Transição de Estados do Processo Cliente 162
A.2 Diagrama de Transição de Estados do Processo Servidor 170
A.3 Diagrama de Transição de Estados do Meta-servidor 170

Anexo B: Cenários para a simulação do Sistema de Arquivos Descentralizado 171
B.1 Cenários para a operação de Abertura de Arquivo 171
B.2 Cenários para a operação de Fechamento de Arquivo 173
B.3 Cenários para a operação de Remoção de Arquivo 179
B.4 Cenários para a operação de Renomeação de Arquivo 185
B.5 Cenários para a Escrita no Arquivo 197
B.6 Cenários para a Leitura do Arquivo 198

Anexo C: Especificação Formal do Sistema Descentralizado 200

LISTA DE FIGURAS

Figura 1: Segmentação e distribuição de dados num arquivo paralelo 5
Figura 2: Arquitetura de um sistema de arquivos paralelos distribuídos 8
Figura 3: Exemplo de uma estrutura de processos 17
Figura 4: Representação de um sistema em SDL 37
Figura 5: Refinamento do bloco b do sistema S 38
Figura 6: Representação gráfica e textual de estados e transições 43
Figura 7: Representação de um sistema em Estelle 46
Figura 8: Refinamento do módulo b do sistema S 46
Figura 9: Comparação entre TDFs 60
Figura 10: Arquitetura de um sistema de arquivos paralelos distribuídos 70
Figura 11: Arquitetura do Framework Conceitual 74
Figura 12: Arquitetura Estelle de um sistema de arquivos paralelos distribuídos 78
Figura 13: Esqueleto da especificação Estelle da arquitetura de um sistema 80
Figura 14: Refinamento dos módulos Cliente e Servidor 81
Figura 15: Esqueleto da especificação do corpo de Cliente 82
Figura 16: Esqueleto da especificação do corpo de Servidor 83
Figura 17: Diagrama de aninhamento de unidades de comportamento 86
Figura 18: Tabela de estados para um comportamento 87
Figura 19: Interação entre processos durante abertura de arquivo 95
Figura 20: MEFE para o processo Cliente 98
Figura 21: MEFE para o processo Servidor 100
Figura 22: MEFE para um Meta-servidor 100
Figura 23: Tabela de Estados do processo Servidor 101
Figura 24: Interações da Tabela de Estados do Servidor 101
Figura 25: Cenários da tentativa bem sucedida de abertura de arquivo 109
Figura 26: Cenários da tentativa mal sucedida de abertura de arquivo 112
Figura 27: Posição Inicial de E/S de Dados 115
Figura 28: Número de Stripes 116
Figura 29: Posição Final de E/S de Dados 116
Figura 30: Situações para as operações de leitura e escrita 117
Figura 31: Arquitetura de um sistema de arquivos paralelos centralizado 120
Figura 32: Interações entre processos durante a abertura de arquivo 121
Figura 33: MEFE para o processo Mestre 123
Figura 34: MEFE para o processo Cliente 124
Figura 35: Arquitetura Estelle de um sistema de arquivos centralizado 126
Figura 36: Esqueleto da especificação centralizada 127
Figura 37: Refinamentos dos módulos Cliente, Servidor e Mestre 128
Figura 38: Esqueleto da especificação do corpo de Mestre 129
Figura 39: Cenários da tentativa bem sucedida de abertura de arquivo 132
Figura 40: Cenários da tentativa mal sucedida de abertura de arquivo 133
Figura 41: Modificações na arquitetura inicial 137
Figura 42: Divisão da especificação inicial 138
Figura 43: Etapas para a implementação de um sistema distribuído 138
Figura 44: Implementação de função primitive 141
Figura 45: Definição da estação e da porta para o processo Mestre 142
Figura 46: Situações para simulação distribuída 145

Figura 47: Escrita de dados para o sistema implementado semi-automaticamente 149
Figura 48: Leitura de dados para o sistema implementado semi-automaticamente 149
Figura 49: Escrita de dados para o sistema implementado manualmente 149
Figura 50: Leitura de dados para o sistema implementado manualmente..... 149

LISTA DE TABELAS

Tabela 1: Cenários de Simulação do sistema descentralizado 113
Tabela 2: Cenários de Simulação do sistema centralizado..... 134

1 Introdução

O ciclo de vida dos sistemas distribuídos normalmente é constituído das seguintes fases: levantamento de requisitos, especificação, implementação e realização. Associadas a essas fases, técnicas de validação (verificação, simulação e teste) geralmente são empregadas (BOLOGNESI; VAN DE LAGEMAAT; VISSERS, 1995).

A partir dos requisitos desejados para um sistema é fundamental a elaboração de uma especificação clara e concisa do mesmo. A construção desse tipo de especificação, utilizando-se linguagens naturais, não é uma tarefa trivial tendo em vista que tais linguagens são intrinsecamente ambíguas.

Visando viabilizar o desenvolvimento de especificações formais de sistemas distribuídos e protocolos de comunicação, várias Técnicas de Descrição Formais (TDFs) foram propostas nos últimos anos, dentre as quais se destacam *Extended State Transition Language (Estelle)* (ISO, 1989a), *Language of Temporal Ordering Specification (LOTOS)* (ISO, 1989b), *Enhancements to LOTOS* (*E-LOTOS*) (ISO, 2001) padronizadas pela *International Organization for Standardization (ISO)*, e *Specification and Description Language (SDL)* (ITU-T, 1996), padronizada pela *International Telecommunications Union - Telecommunication (ITU-T)*.

Para dar suporte à utilização dessas TDFs, foram construídas várias ferramentas que auxiliam o projetista tanto na edição, verificação e simulação de especificações formais de sistemas, quanto na implementação e teste desses sistemas a partir de suas especificações.

Este projeto de pesquisa tem por objetivo o desenvolvimento de uma abordagem, baseada na utilização de *frameworks*, TDFs e um conjunto de ferramentas, para a especificação, implementação e validação de sistemas de arquivos paralelos distribuídos.

1.1 Justificativa

O constante aumento da velocidade de processamento tem propiciado grandes avanços no projeto e construção de sistemas computacionais paralelos. Esse aumento

deve-se sobretudo à utilização de um número cada vez maior de processadores nessas arquiteturas. Entretanto, o desempenho de muitas aplicações paralelas tem sido afetado principalmente devido às operações de entrada e saída (E/S) de dados. Isto porque aplicações paralelas, com uma alta demanda de processamento, também realizam um elevado número de operações de acesso e armazenamento temporário de informações, o que faz das operações de E/S o maior gargalo para aplicações dessa categoria (DEL ROSARIO; CHOUDHARY, 1994; POOLE, 1994).

Assim como no caso da demanda por processamento, o paralelismo oferece uma solução para que as limitações no desempenho individual de cada disco sejam superadas. Unidades de armazenamento especiais utilizam diversos discos para o aumento da taxa de transferência nas operações de E/S de dados. De uma forma geral esses dispositivos procuram particionar e distribuir os dados dos arquivos entre diversos discos, dando origem aos arquivos paralelos. No entanto as altas taxas de transferência, oferecidas pelos dispositivos de armazenamento paralelo, não eliminam o gargalo inerente a um sistema de arquivos centralizado, seqüencial ou manipulado por uma única aplicação. Como decorrência, muitos sistemas de arquivos continuam a apresentar baixo desempenho, mesmo com a utilização de dispositivos paralelos de armazenamento.

Buscando melhorar a performance de tais sistemas, indústria e comunidade acadêmica têm devotado esforços objetivando tornar as operações de E/S mais eficientes, melhorando aspectos como a taxa de transferência e a latência no acesso aos dados. Diversas propostas, para novos sistemas de arquivos paralelos, têm surgido, visando oferecer suporte para as aplicações que necessitam de E/S de alto desempenho. De maneira geral, tais sistemas procuram combinar as taxas de transferência, no acesso em paralelo, a diversos discos (simples ou paralelos) para o aumento do *throughput* nas transmissões entre discos e memória.

O princípio básico desses sistemas é a fragmentação dos arquivos e a distribuição dos blocos resultantes em diversos servidores, conectados por meio de uma rede de comunicação. Essa estruturação possibilita o acesso simultâneo aos fragmentos por parte dos processos paralelos, que realizam, através de um conjunto de primitivas, operações semelhantes àquelas encontradas em servidores de arquivos centralizados. O surgimento desses sistemas levou ao desenvolvimento de novos compiladores, linguagens de programação e sistemas de arquivos, que hoje são usados na implementação de aplicações paralelas, visto que estas apresentavam baixo

desempenho, quando executados, por exemplo, em sistemas de E/S baseados no Unix, desprovidos de um controle refinado em relação à organização de arquivos e serviços oferecidos.

Para o projeto e implementação de um sistema de arquivos paralelos distribuídos, a utilização de métodos tradicionais informais (ou semiformais) de descrição, tais como as tabelas de estados, não garante que todos os aspectos do sistema estejam presentes na especificação e nem que estejam descritos de forma correta e não ambígua. Possíveis erros presentes nessa especificação informal (ou semiformal) geralmente são detectados através de testes, realizados após a implementação do sistema. O emprego de uma TDF, para a especificação formal desse tipo de sistema, além de produzir uma descrição clara, concisa e livre de ambigüidades, oferece ao projetista a possibilidade de verificar, com o auxílio de ferramentas apropriadas, a correção do sistema antes mesmo que sua implementação esteja concretizada. É neste contexto que se enquadra esta pesquisa, na medida em que propõe uma abordagem, baseada em métodos formais, para o projeto e implementação de tais sistemas.

1.2 Sistemas de Arquivos Paralelos Distribuídos

Embora existam sistemas de arquivos paralelos distintos, todos possuem o mesmo princípio básico, que é a fragmentação dos arquivos e a distribuição dos blocos resultantes em diversos servidores conectados entre si por meio de uma rede de comunicação. Essa estruturação possibilita o acesso simultâneo aos fragmentos por parte dos processos paralelos, que realizam, através de um conjunto de primitivas, operações semelhantes àquelas encontradas em servidores de arquivos centralizados.

O principal diferencial entre arquivos paralelos e arquivos comuns é a necessidade, nos primeiros, de uma alta taxa de transferência de dados associada à capacidade de acesso concorrente aos arquivos por parte dos processos. Caracteriza-se, assim, como entrada e saída paralela o suporte provido por uma aplicação, executada em diversos nós, que permite aos dados das aplicações, que se encontram fisicamente distribuídos, sejam manipulados como se fossem um único arquivo lógico denominado arquivo paralelo.

Um sistema de arquivos paralelos tem como objetivo realizar operações sobre arquivos de forma transparente aos seus processos Usuários. A fragmentação do arquivo fica, então, oculta, fornecendo uma visão global ao Usuário que vê o arquivo distribuído como uma única estrutura lógica. No entanto, a fim de otimizar operações sobre um arquivo, os Usuários do sistema podem fazer uso de sua visão interna, que revela a estruturação desse arquivo nos discos.

1.2.1 Distribuição dos Dados

Um arquivo paralelo corresponde a um único arquivo lógico, composto de um ou mais blocos de dados fisicamente disjuntos. Na criação de um arquivo como esse, os servidores que irão acomodar seus fragmentos devem ser identificados numa operação denominada como *clustering* ou *declustering*. A partir dessa identificação, os dados podem ser atribuídos aos blocos localizados nos servidores correspondentes, o que define um *padrão de distribuição*. Um padrão comum de distribuição é o chamado *striping*. Nesse caso, as unidades de distribuição, cujo tamanho pode variar de 1 bit ao tamanho do bloco lógico utilizado na formatação do disco, são atribuídas aos discos de forma circular (*round-robin*). O conjunto de dados atribuídos a todos os discos numa rodada de distribuição constitui um *stripe*.

Cada bloco de dados armazenado de maneira contígua num disco define um segmento, caracterizado por um conjunto de unidades de distribuição (Figura 1) (GUARDIA, 1999). O tamanho das unidades de distribuição é sempre o mesmo para um determinado arquivo, objetivando facilitar a manipulação do arquivo distribuído. A mesma regra pode ser aplicada sempre que possível ao tamanho dos segmentos, que excepcionalmente pode variar na presença de um segmento incompleto no fim do arquivo.

A granularidade das unidades de distribuição tem influência direta no desempenho das aplicações. Para um balanceamento de carga mais eficiente entre os servidores do sistema, a granularidade fina pode ser adotada. Nesse caso, os discos armazenam apenas uma fração de um bloco que pode ser manipulado. Entretanto, a transferência de dados ocorre em conjuntos bastante pequenos, o que pode levar a uma sobrecarga na rede.

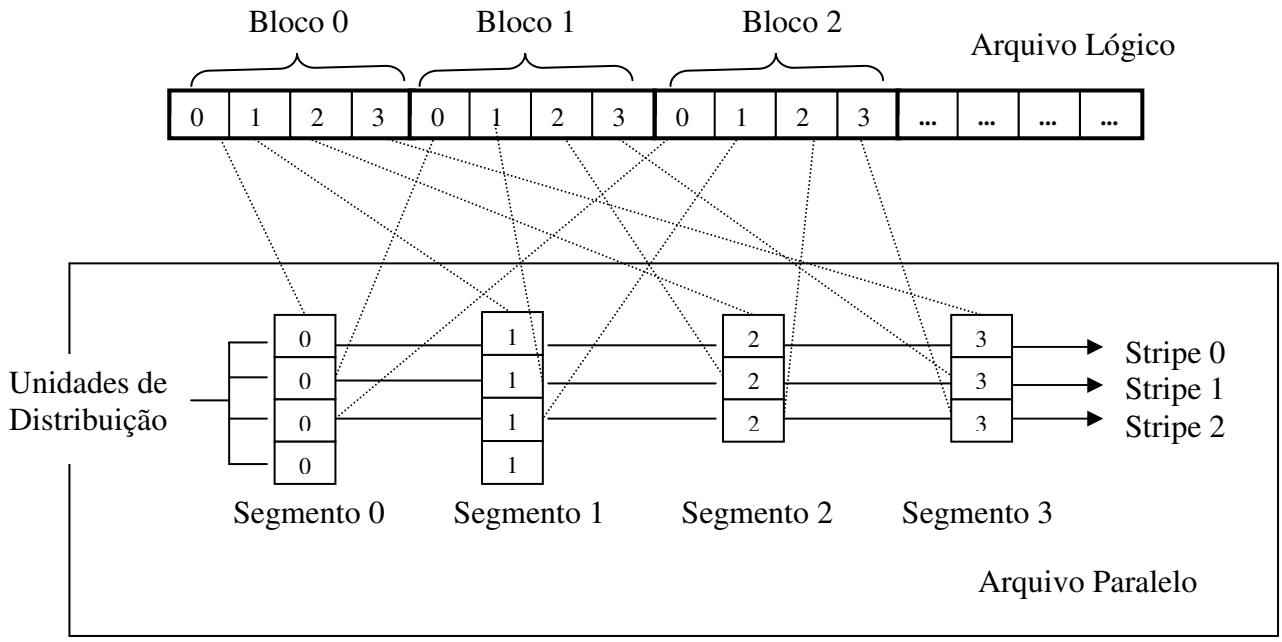


Figura 1: Segmentação e distribuição de dados num arquivo paralelo

Para uma transferência de dados mais otimizada, a granularidade grossa deve ser adotada. O conjunto de dados, na maioria dos casos, atinge um tamanho que é múltiplo do tamanho dos blocos do disco. O paralelismo na transferência é melhor explorado nesse caso, e o atendimento de pequenas requisições pode ser feito de maneira concorrente por discos independentes.

1.2.2 Organização de Arquivos Paralelos e Primitivas de Acesso

O problema de E/S pode ser abordado inicialmente através da identificação da função que os arquivos representam para as aplicações. Dessa maneira, é possível determinar diferentes formas de organização que visam atender às necessidades dessas aplicações.

No processamento paralelo, um problema grande e complexo é particionado em vários subproblemas que podem ser executados simultaneamente mantendo um certo grau de interação entre si. Do mesmo modo, os dados do problema global também devem ser divididos em subconjuntos para que possam ser atribuídos aos diversos processos ou processadores.

Embora cada aplicação exija uma maneira distinta de realizar essa partição, podem ser observados alguns esquemas comuns para a divisão dos dados nos discos (GUARDIA, 1999). Como exemplo, se os dados tratados por uma aplicação estão organizados estruturalmente como matrizes, sua distribuição pode seguir um padrão de linhas, colunas ou submatrizes. Os blocos formados nos arquivos por esses dados podem ser associados a um ou mais processos e acessados de maneira seqüencial ou aleatória.

Os sistemas de arquivos paralelos distribuídos buscam fornecer um conjunto de primitivas para acesso aos arquivos com uma sintaxe a mais parecida possível com aquela encontrada em sistemas de arquivos locais e centralizados. Primitivas como *open*, *close*, *seek*, *rename* e *unlink*, para abrir, fechar, alterar a posição do ponteiro de arquivo, renomear e remover arquivos, respectivamente, são disponibilizadas por esses sistemas. Operações de E/S podem ser realizadas pelos processos através da chamada das primitivas *read* e *write*.

Os parâmetros dessas funções são semelhantes aos de funções equivalentes para arquivos comuns. Entretanto, parâmetros extras geralmente são adicionados para indicar o número de servidores envolvidos com um arquivo, o tamanho dos blocos nas partições e o padrão de distribuição utilizado, apenas para citar alguns exemplos.

Mesmo havendo características diferentes entre sistemas de arquivos paralelos, os fabricantes buscam adotar uma interface comum aos seus sistemas, como o padrão proposto pelo projeto *Scalable I/O* (*SIO*) (BERSHAD, 1994; CHOUDHARY et al., 1994; BAGRODIA et al., 1994) que visa deixar a manipulação de arquivos a mais transparente possível aos processos.

Mesmo que seja oferecida a transparência, muitas vezes é interessante que o programador de uma aplicação distribuída tenha conhecimento do padrão de acesso que o sistema de arquivos utiliza, o que permite desenvolver aplicações otimizadas para um determinado tipo de sistema e configuração. Operações como *prefetching*, que é a busca antecipada de informações nos discos, e a redução de latência através do uso de *cache*, que mantém os dados mais solicitados na memória, se beneficiam de uma aplicação estruturalmente devotada às características do sistema de arquivos sobre o qual é executada.

1.2.3 Arquitetura dos Sistemas de Arquivos Paralelos Distribuídos

Ao ser projetado um sistema operacional, um sistema de arquivos paralelos pode ser desenvolvido de forma a já considerar a existência de diversos discos para a distribuição de dados. Exemplos desses sistemas são o *Parallel Input/Output File System* (*PIOFS*) (CORBETT et al., 1995) e o *Parallel File System* (*PFS*) (BERSHAD, 1994).

Em sistemas operacionais que não possuem sistemas de arquivos paralelos nativos, o paralelismo pode ser implementado por meio de camadas de software que atuam junto aos Usuários e servidores do sistema destinados ao armazenamento de arquivos distribuídos em seus discos locais.

O estudo dos diversos tipos de sistemas de arquivos paralelos distribuídos já implementados, revela que estes compartilham diversas características estruturais. Pode-se identificar dois tipos genéricos de arquitetura para esses sistemas, conforme realizam algumas operações sobre arquivos e operações de gerenciamento de forma descentralizada ou centralizada. A arquitetura mais simples é a descentralizada, que define apenas dois tipos de processos que, trabalhando em conjunto, fornecerão as funcionalidades de um sistema de arquivos paralelos sobre um sistema de arquivos convencional. Na arquitetura centralizada, além dos processos junto aos Usuários e servidores, também é considerada a existência de um processo coordenador centralizado. A Figura 2 demonstra as similaridades entre os dois tipos de arquitetura, com o processo coordenador representado em destaque.

Nos dois tipos de arquitetura o Cliente é um processo do Usuário que utiliza um conjunto de primitivas disponíveis para acesso aos dados distribuídos. As primitivas mais básicas tendem a seguir a sintaxe Unix para manipulação de arquivos. Nos servidores o acesso aos fragmentos dos arquivos armazenados é feito pelo processo Servidor, que faz com que esses fragmentos sejam tratados pelo sistema de arquivos local como se fossem arquivos comuns.

Na arquitetura descentralizada, todas as operações são realizadas diretamente entre os processos Cliente e Servidor sem nenhuma intermediação. Os meta-dados dos arquivos, que incluem informações sobre o método de distribuição utilizado para um determinado arquivo, tamanho da unidade de distribuição, servidores envolvidos, o estado atual de cada arquivo, entre outras, ficam distribuídos entre os próprios

servidores do sistema. Essas informações são consultadas pelos Clientes para verificar se uma operação solicitada por seus Usuários sobre um arquivo distribuído compartilhado é coerente dentro da semântica de grupo. Exemplo de uma operação incoerente é a tentativa de um Usuário de remover um arquivo aberto por outro Usuário.

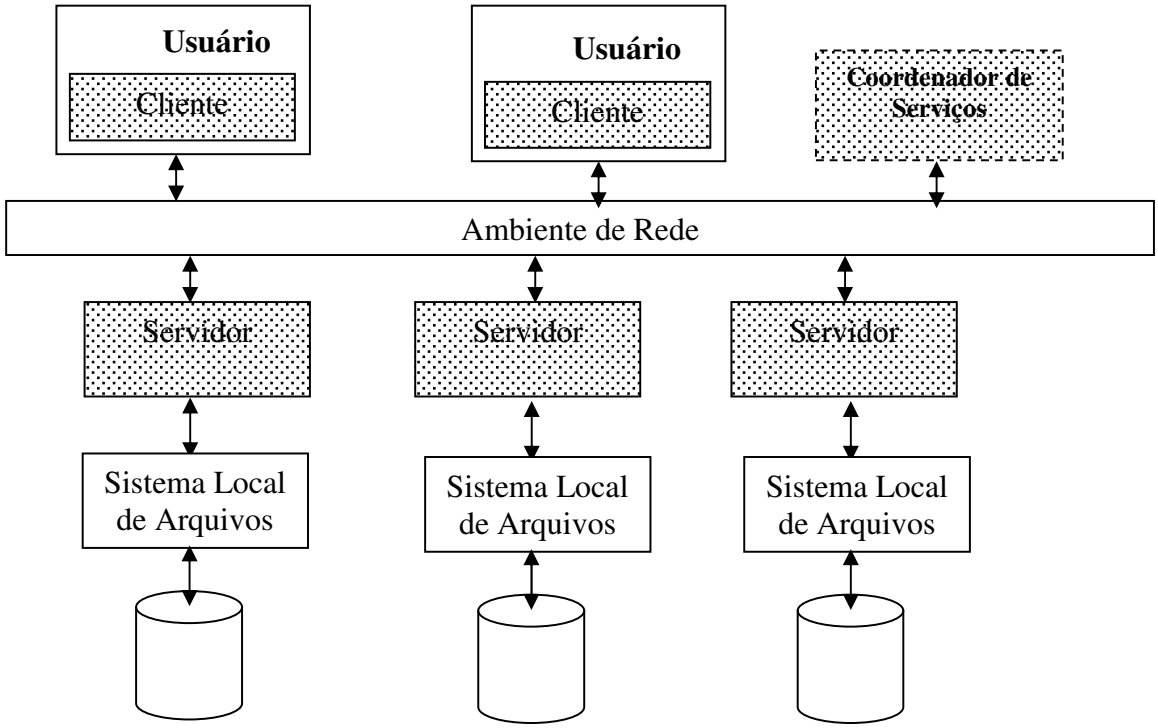


Figura 2: Arquitetura de um sistema de arquivos paralelos distribuídos

Na arquitetura com o processo *Coordenador de Serviços*, este é encarregado pelo armazenamento dos meta-dados, pela manutenção dos servidores e gerenciamento de operações centralizadas sobre arquivos. Operações como abertura, fechamento, remoção e renomeação são feitas por intermédio do coordenador, que se encarrega de consultar os servidores envolvidos com um arquivo e de retornar o resultado da operação ao Usuário que solicitou o serviço. Para aumentar o desempenho, operações de leitura e escrita de dados nos arquivos distribuídos são feitas diretamente entre Clientes e Servidores.

Embora seja a mais comum, nem todos os sistemas de arquivos paralelos seguem fielmente essa arquitetura. Dependendo das características de cada um, novos processos podem ser adicionados ou suprimidos do sistema.

Exemplos de sistema que não possuem um processo coordenador de serviços incluem o *Expand*, *SPFS* e *Galley*.

O *Expand* (GARCIA et al., 2002) é um sistema que utiliza múltiplos Servidores *NFS* como um único sistema de arquivos, o que possibilita a distribuição de arquivos paralelos sobre uma arquitetura heterogênea de computadores. Todas as funcionalidades são implementadas nos Clientes, dispensando a utilização de um processo Coordenador e de processos atuando nos Servidores.

As informações (meta-dados) sobre um arquivo são armazenadas em um dos nós Servidores onde estão distribuídos os segmentos desse arquivo, denominado de nó Mestre. Para encontrar o nó Mestre, os Clientes utilizam uma função *hash* que, dado o nome de um arquivo, retorna um número inteiro que identifica o Servidor onde o nó está localizado. Essa abordagem aumenta a confiabilidade do sistema ao descentralizar o registro de meta-dados, porém cria novos problemas. Por exemplo, a renomeação de um arquivo altera o valor retornado pela função *hash*, exigindo que o nó Mestre seja removido para um novo Servidor.

O sistema *Scotch Parallel File System* (*SPFS*) (GIBSON et al., 1995) compartilha algumas características do *Expand*, como a distribuição dos meta-dados entre os Servidores. Objetiva uma melhor utilização do sistema de redundância do *RAID*, permitindo que diversas requisições pequenas de escrita seqüenciais tenham sua paridade calculada em conjunto ao invés de individualmente. Esse cálculo adiado é denominado de *ponto de paridade*.

Para aumentar o desempenho, o *SPFS* utiliza uma política agressiva de busca antecipada de dados e de escrita adiada através do armazenamento de dados nos *buffers* locais dos Clientes. Quando um Cliente atualiza os dados em seu buffer, este deve propagar explicitamente essa atualização aos demais Clientes para evitar inconsistência dos dados. Assim sendo, o *SPFS* implementa uma forma de memória compartilhada fracamente consistente.

O *Galley* (NIEUWEJARR; KOTZ, 1996) é um sistema que possui em sua arquitetura processos Clientes (*Computer Processors – CP*) e Servidores (*I/O Processors – IOP*). A função dos Clientes é permitir o acesso pelos aplicativos aos

arquivos distribuídos entre os diversos Servidores. Assim como ocorre nos sistemas *Expand* e *SPFS*, os Servidores ficam incumbidos de armazenar e gerenciar os meta-dados.

O sistema apresenta uma interface simples e genérica, que possibilita aos aplicativos controlar de forma explícita o paralelismo no acesso aos arquivos. Dessa maneira, sem atender diretamente a necessidade de cada Usuário, *Galley* fornece suporte a uma ampla variedade de bibliotecas construídas para de adequar a uma categoria particular de aplicação.

Exemplos de sistemas que incluem um processo coordenador de serviços ou mesmo outros processos são o *NPFS*, *PIOUS*, *PPFS*, *PPFS II* e *Zebra*.

A estrutura do *Network Parallel File System* (*NPFS*) (GUARDIA; SATO, 1999) consiste de um conjunto de Servidores, de um coordenador, denominado Mestre, e de rotinas de bibliotecas que são incluídas nos processos Usuários. A troca de informações entre os processos Clientes e os componentes da arquitetura é realizada utilizando um serviço de comunicação disponível no protocolo de transporte.

Além de prover armazenamento e recuperação de dados de alto desempenho, sua funcionalidade consiste em servir como ferramenta de teste para o estudo de otimizações aplicáveis na execução de aplicações distribuídas. O *NPFS* possibilita o conhecimento sobre a localização dos arquivos, permitindo uma distribuição mais balanceada entre os dados armazenados nos discos e as aplicações que os manipulam.

O *Parallel Input/OUtptut System* (*PIOUS*) (MOYER; SUNDERAN, 1994) possui uma estrutura análoga ao do *NPFS*. É formado por um conjunto de Servidores de dados (*PIOUS Data Servers – PDS*) e por um coordenador de serviços (*PIOUS Service Coordinator – PSC*), além dos processos Clientes. Os arquivos distribuídos são denominados *parafiles* e podem ser acessados pelos Usuários de forma global ou fragmentada.

PIOUS possui dois grupos distintos de transações: estável (*stable*) e volátil (*volatile*). As transações estáveis asseguram tolerância a falhas e são as preferencialmente empregadas pelas aplicações. As voláteis não possuem essa proteção,

podendo porém render um melhor desempenho de E/S. A distribuição de dados nos arquivos é feita de forma transparente aos Usuários.

O *Portable Parallel File System* (*PPFS*) (HUBER et al., 1995) é um sistema idealizado para ser executado sobre estações Unix com o apoio de mecanismos de comunicação por mensagens como *MPI*. Possui uma interface de programação flexível, através da qual uma aplicação pode controlar ou mesmo criar políticas de E/S que melhor atendam suas necessidades. Sua arquitetura é semelhante ao do *NPFS* e *PIOUS*, com Clientes, servidores e um administrador de meta-dados. Para otimizar o acesso e a busca antecipada de dados de um arquivo compartilhado, o *PPFS* utiliza *agentes de cache* que são compartilhados por diversos Clientes que acessam o mesmo arquivo distribuído.

PPFS II é um sistema de arquivos paralelos adaptativo que evoluiu a partir do *PPFS*. Sua principal contribuição é a introdução de um sistema que monitora, em tempo real, a interação de uma aplicação com o sistema de arquivos paralelos, visando escolher dinamicamente a melhor política de acesso aos dados. O *PPFS II* foi projetado tendo em vista que algumas aplicações apresentam padrões irregulares de E/S durante a execução, o que exigiria a alteração dinâmica das configurações do sistema de arquivos.

Zebra (HARTMAN; OUSTERHOULT, 1993) é um sistema idealizado para particionar arquivos de diversos tamanhos de uma maneira mais eficiente, facilitando o uso dos mecanismos de paridade presentes nos conjuntos de discos *RAID*. A partição convencional por arquivo (*per-file striping*) mostrou-se ineficiente ao lidar com arquivos pequenos os quais, devido ao tamanho reduzido dos fragmentos distribuídos que geram, pouco se beneficiam do paralelismo. A solução seria armazenar arquivos pequenos em um único Servidor. Entretanto isso leva ao uso ineficiente do disco de paridade, já que o espaço reservado para a paridade de um arquivo pequeno ou de um grande é o mesmo.

Para contornar essas limitações, *Zebra* utiliza o particionamento por Cliente (*per-client striping*) utilizando técnicas de sistemas de arquivos baseados em registros de operações (*Log-structured File Systems - LFS*). Nessa abordagem as requisições de escrita dos Clientes são agrupadas em grandes blocos (*logs*) que, por sua vez, são

particionados e distribuídos entre os Servidores. Um mesmo bloco pode comportar diversos arquivos pequenos que são escritos nos discos numa única transferência.

Os blocos somente podem ser anexados no fim do disco lógico, o que facilita o cálculo da paridade. No entanto, caso um arquivo precise ser atualizado, seus novos dados só poderão ser escritos num novo bloco a ser incluído no fim da estrutura de blocos, deixando assim um bloco de dados desatualizado que precisa ser descartado e reutilizado. Por essa razão, além dos Clientes, do Controlador de Arquivos (*File Manager*) e dos Servidores (*Storage Servers*), *Zebra* também inclui na arquitetura da Figura 2 um módulo encarregado da reutilização dos blocos desatualizados (*Stripe Cleaner*).

1.3 Metodologia

Inicialmente foi realizada uma extensa pesquisa bibliográfica relativa aos sistemas de arquivos paralelos distribuídos, a fim de determinar as suas principais características. Paralelamente, foi realizado um estudo das principais TDFs, considerando os conceitos arquitetônicos, as sintaxes e as semânticas envolvidas e as ferramentas disponíveis, visando a escolha da TDF mais adequada à especificação de tais sistemas.

Uma vez encerrados os estudos foi definida uma arquitetura genérica, para um sistema de arquivos paralelos distribuídos, que englobasse os requisitos compartilhados por esses sistemas. A fim de tornar a abordagem proposta o mais abrangente possível, essa arquitetura foi construída utilizando-se o conceito de *framework*.

Frameworks são estruturas de *software* desenvolvidas para um domínio de aplicação específico, que podem ser reutilizadas na implementação de diferentes sistemas pertencentes a esse domínio. Para o seu emprego, um sistema distribuído deve ser particionado em componentes que apresentam um comportamento interno específico e que se comunicam através de interações. Esse procedimento é o mesmo utilizado no projeto de sistemas distribuídos a serem especificados através de TDFs.

Conseqüentemente a adoção de *frameworks* neste projeto permite a criação de uma especificação formal de um modelo básico para sistemas de arquivos distribuídos, onde as especificidades de cada sistema podem ser posteriormente introduzidas a esse

modelo básico, levando a uma adaptação da arquitetura inicial em conformidade com os requisitos desses sistemas. Embora elementar, a arquitetura desse modelo básico define um sistema funcional, sobre a qual podem ser adicionados, por exemplo, novos componentes e operações sobre arquivos.

A escolha da TDF mais apropriada para a concepção do *framework* é orientada através do estudo de um meta-modelo, que reúne as principais características estruturais e comportamentais empregadas no projeto desse tipo de sistema. A similaridade dessas características com as oferecidas pelas TDFs é analisada, oferecendo os meios para decidir qual TDF melhor se aplica na construção do *framework*.

Definida a TDF, foi especificada a arquitetura para um sistema padrão utilizando-se os conceitos dessa TDF. Visando o estudo de casos, a partir dessa arquitetura foram derivadas as especificações formais de dois sistemas de arquivos. O primeiro obedecendo a arquitetura do *framework* e o segundo adicionando a esta um novo componente, a fim de demonstrar a capacidade de adaptação do *framework* a novos requisitos. Os sistemas foram posteriormente submetidos à validação com auxílio de um simulador, buscando-se detectar erros nas especificações e avaliar determinadas propriedades requeridas aos sistemas.

A partir da especificação formal validada, a implementação, numa determinada linguagem de programação, do sistema mais elaborado, que dispõe de um maior número de componentes, pôde ser gerada semi-automaticamente com o auxílio de um compilador relativo à TDF escolhida. As partes dessa implementação, dependentes do ambiente operacional e dos mecanismos de comunicação, foram codificadas manualmente.

Para os testes de conformidade, as seqüências de teste foram obtidas a partir da especificação formal sendo em seguida aplicadas à implementação. Os traços obtidos foram também analisados com base nessa especificação. Para a realização do sistema, isto é, a implantação da implementação num ambiente distribuído específico, e para os testes desse sistema, uma rede de computadores foi utilizada. Estudos comparativos de desempenho, entre um sistema desenvolvido segundo a abordagem proposta neste projeto e outro equivalente codificado manualmente, forneceram subsídios para a avaliar a aplicabilidade dessa abordagem.

1.4 Organização do Texto

O Capítulo 2 fornece uma visão das principais TDFs, enfocando seus conceitos e construções, e as principais ferramentas construídas para essas TDFs. O Capítulo 3 descreve a abordagem proposta nesse projeto, sendo que um *framework* conceitual para um sistema de arquivos distribuídos básico é desenvolvido. O Capítulo 4 apresenta um estudo de caso para a aplicação da abordagem descrita no capítulo anterior que segue a arquitetura do *framework*. O Capítulo 5 demonstra um segundo estudo de caso em que há a necessidade de alterações na arquitetura básica. O Capítulo 6 apresenta a implementação de um dos sistemas especificados como estudo de caso, encerrando o ciclo de desenvolvimento de tais sistemas no contexto da abordagem proposta. Finalmente o Capítulo 7 tece as conclusões desse projeto e aponta algumas direções para trabalhos futuros.

2 Técnicas de Descrição Formal

A descrição de um sistema pode ser realizada informalmente através de linguagens naturais associadas ou não a tabelas de estados ou diagramas de transições. Descrições obtidas dessa maneira tendem a ser ambíguas e de difícil análise, abrindo a possibilidade de presença de erros e omissões no projeto do sistema descrito. Falhas em sistemas complexos não são facilmente detectáveis e possuem um alto custo de correção após a implementação do sistema.

Para conferir uma maior precisão e confiabilidade ao desenvolvimento de sistemas distribuídos, *Técnicas de Descrição Formal (TDFs)* foram elaboradas. Resultado de várias décadas de trabalho sobre métodos e linguagens de descrição formal, as TDFs possuem um embasamento matemático que assegura a descrição de um sistema de uma maneira precisa, a qual pode ser submetida à análise e à validação das propriedades requeridas.

Assim sendo, a especificação de um sistema através de uma TDF deve apresentar as seguintes propriedades:

- ser clara, concisa e sem ambigüidades;
- ser completa, sem omitir nenhum detalhe do sistema;
- ser consistente, ou seja, a especificação deve refletir com fidelidade todas as características do sistema. Especificações do mesmo sistema em diferentes níveis de abstração devem ser equivalentes;
- ser tratável, o que significa que a especificação pode ser submetida à análise para verificação e validação de suas propriedades;

Um dos benefícios do uso de uma TDF é a descoberta de erros, ambigüidades e inconsistências antes da fase de implementação. Problemas de estruturação também podem ser detectados e corrigidos logo nos primeiros estágios do projeto. Mesmo que já se tenha um desenvolvimento informal num grau avançado, a aplicação de uma TDF pode ainda ser útil para a detecção de deficiências desse projeto.

Atualmente há diversos padrões de TDFs, dentre as quais destacam-se Estelle, LOTOS e E-LOTOS, padronizadas pela ISO, e SDL, padronizada pela ITU-T. Essas

TDFs compartilham uma base comum para especificação de comportamento, chamada de sistema de transições rotuladas, onde as transições entre estados são rotuladas com ações associadas. Para a tipificação de dados, Estelle usa os tipos de dados de Pascal, enquanto que as demais TDFs usam Tipos Abstratos de Dados, que são definidos algebricamente.

2.1 LOTOS

Language Of Temporal Ordering Specification (*LOTOS*) foi desenvolvida para definir padrões formais de serviços e protocolos OSI independentes de implementação. LOTOS possui duas partes claramente separadas. A primeira parte fornece um modelo comportamental derivado de álgebras de processos, principalmente de *Calculus of Communicating Systems* (*CCS*) (MILNER, 1989) e de *Communicating SequentialProcesses* (*CSP*) (HOARE, 1985). A segunda parte, baseada na linguagem *Algebraic Specification Techniques for Correct and Trusty Software Systems 1* (*ACT ONE*) (EHRIG, 1985), permite aos especificadores descrever tipos e valores abstratos de dados.

Esta seção fornece uma visão geral dos principais aspectos de LOTOS. Para uma abordagem mais detalhada, recomenda-se consultar os tutoriais (TURNER, 1996; VAN EIJK, 1989; TURNER, 1993).

2.1.1 Conceitos Básicos

Para a modelagem de sistemas, LOTOS utiliza os conceitos básicos de processo, evento e expressão de comportamento.

Sistemas e seus componentes são representados como *processos*. Um processo é representado como uma caixa-preta, cujo comportamento é descrito por um observador externo, definindo-se uma relação temporal dos eventos observáveis. Processos podem ser refinados em subprocessos, permitindo a representação de um sistema e de suas

partes. A Figura 3 ilustra a estrutura de um sistema S em LOTOS e o refinamento deste sistema em dois subprocessos P e Q.

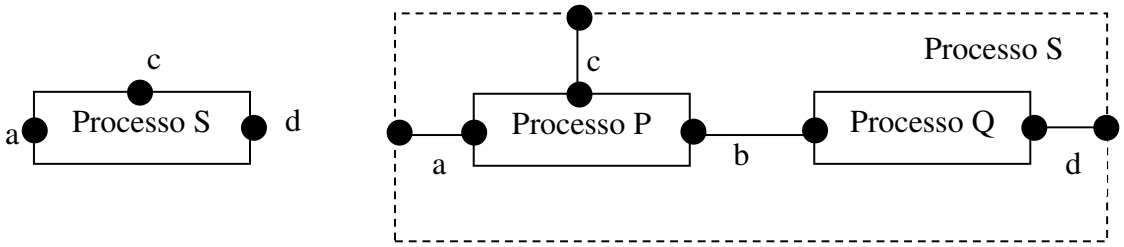


Figura 3: Exemplo de uma estrutura de processos

A interação de um processo com seu ambiente ocorre através de uma *porta*. O processo *S* comunica-se com o seu ambiente através das portas *a*, *c* e *d*. O processo *P* comunica-se com seu ambiente através de *a* e *c*. O processo Q, por sua vez, faz a comunicação com o ambiente através de *d*. Esses dois processos se comunicam pela porta *b*.

Textualmente a definição do sistema S, correspondente à Figura 3, ficaria da seguinte forma:

```
process S [a, c, d]: noexit :=
  hide b in
    P [a, b, c] | [b] | Q [b, d]   (* expressão de comportamento *)
where
  process P [t, u, v]: noexit :=
    t; (u; stop [] v; stop)        (* expressão de comportamento *)
  endproc (* P *)
  process Q [x, y]: noexit :=
    x; y; stop                     (* expressão de comportamento *)
  endproc (* Q *)
endproc (* S *)
```

Declarações de processos são delimitadas pelas palavras-chaves *process* e *end-proc*. As definições de processos possuem um identificador de processo, uma lista de portas, uma lista opcional de parâmetros, a funcionalidade do processo e expressões de comportamento. Em LOTOS uma *especificação* é um tipo especial de processo que representa todo o sistema.

No exemplo, P[a,b,c] e Q[b,d] representam as instanciações dos processos P e Q respectivamente. A lista que acompanha o nome do processo representa a lista de identificadores de portas. O operador |b| entre P e Q afirma que esses processos interagem através da porta comum *b*, que é oculta (*hide*) em relação ao ambiente. Um mesmo processo pode ser instanciado diversas vezes por outros processos ou por ele mesmo (instanciação recursiva de processo), o que permite representar comportamentos repetitivos ou mesmo infinitos. A funcionalidade do processo é definida pela palavra-chave *exit* para processos que possuem um término e por *noexit* para aqueles que nunca terminam, como processos recursivos que ficam executando indefinidamente.

As interações entre processos são representadas por *eventos*. Um evento é atômico, instantâneo e síncrono, associado a uma porta com o mesmo nome do evento. Um evento ocorre somente se todos os processos que participam dele estiverem prontos para interagir. Na sua ocorrência todos os processos envolvidos são sincronizados, passando a compartilhar os mesmos valores de parâmetros da interação.

O comportamento observável de um sistema é descrito em LOTOS por meio de *expressões de comportamento*, que definem as seqüências permitidas de eventos. Por exemplo, na definição do sistema *S* a seqüência de ações

t; (u; **stop** [] v; **stop**)

define uma expressão de comportamento onde ocorre primeiro o evento *t* sendo depois sucedido pelo evento *u* ou *v*. Os construtores que descrevem expressões, como o construtor de escolha de eventos ([]), serão apresentados na próxima seção.

2.1.2 LOTOS Básico

LOTOS sem ACT ONE (utilizado na descrição de tipos de dados) é chamado de LOTOS Básico. Seus processos e operadores básicos permitem a representação de qualquer comportamento finito.

2.1.2.1 Processos e Operadores Básicos

O processo básico *stop*, denominado inação ou ausência de ação, modela uma situação onde não é mais possível a interação com o ambiente e pode ser usado, por exemplo, para descrever impasses (*deadlocks*). Para descrever o encerramento com sucesso é utilizado o processo básico *exit*, cuja execução é modelada através da ocorrência do pseudo evento δ.

O *prefixo de ação* (*;*) é usado para indicar que um evento deve ocorrer antes da próxima expressão de comportamento. Por exemplo, a expressão $a;B_1$ indica que o evento *a* deve ocorrer antes da expressão de comportamento B_1.

A *escolha* (*[]*) permite selecionar um dentre dois comportamentos alternativos. Por exemplo, a expressão B_1 *[]* B_2 se comportará como a expressão B_1 ou como a expressão B_2, dependendo se a próxima ocorrência for o evento inicial de B_1 ou o evento inicial de B_2 respectivamente.

2.1.2.2 Eventos Internos

Até o presente momento o comportamento observável foi descrito em termos de eventos que representam as interações entre um sistema e seu ambiente. Entretanto um sistema pode tomar uma decisão interna não observável que afetará seu comportamento futuro. Em LOTOS o evento $\boldsymbol{i}$ modela ocorrências ou decisões internas e, portanto, invisíveis ao ambiente. Exemplos de tais ocorrências são falhas de sistemas, perda de recursos e *timeout*.

Eventos internos podem levar a um comportamento não determinístico de um sistema. Esse comportamento ocorre quando há múltiplas possibilidades de comportamento sem que o ambiente tenha controle sobre as mesmas. Por exemplo, supondo que *P* e *Q* são expressões de comportamento e que *a* é um evento, na expressão *(a;P) [] (i;Q)* o ambiente poderá influenciar no comportamento se este agir, através da geração do evento *a*, antes que o evento interno *i* ocorra. Já na expressão *(i;P) [] (i;Q)* não há controle algum do ambiente sobre o comportamento, pois nenhuma interação observável influencia sua determinação.

Mesmo que não haja eventos internos, comportamentos não determinísticos podem ocorrer. Por exemplo, na expressão *(a;P) [] (a;Q)* se o ambiente oferece o evento *a*, um dos comportamentos *P* ou *Q* será escolhido indeterministicamente.

2.1.2.3 Composição Paralela e Seqüencial

Geralmente um sistema distribuído é composto de um conjunto de instâncias com funcionalidades paralelas. Essa característica é modelada em LOTOS utilizando-se *construtores paralelos*.

Considere a expressão geral de comportamento $B_1 \,|(g_1,...,g_n)|\, B_2$ onde B_1 e B_2 são expressões de comportamento e $g_1,...,g_n$ é uma lista de identificadores de portas. Nessa expressão de comportamento os eventos pertencentes à lista de portas somente podem ocorrer com a participação conjunta de B_1 e B_2. Os outros eventos de B_1 e B_2, que não pertencem a essa lista, não possuem essa dependência.

Quando a lista de identificadores de portas, que envolve os comportamentos B_1 e B_2, é vazia há o entrelaçamento puro $B_1 \,|||\, B_2$, onde os eventos ocorrem com a participação de apenas uma das expressões de comportamento. Quando a lista de portas possui todos os eventos de B_1 e B_2 há a sincronização plena $B_1 \,||\, B_2$, onde qualquer evento só pode ocorrer com a participação das duas expressões de comportamento. Não há sincronização com eventos internos já que estes não são observáveis.

A representação da ordem temporal de expressões de comportamento, que podem representar fases distintas de um sistema, é realizada em LOTOS com o *construtor de habilitação* >>. Na expressão $B_1 >> B_2$ o comportamento B_2 é habilitado pelo comportamento B_1 se este último terminar com sucesso, isto é, a ocorrência do pseudo evento δ em B_1 habilita B_2.

A interrupção da execução de um comportamento, causada por alguma circunstância excepcional (e.g., ocorrência de um erro), é representada em LOTOS pelo *construtor de desabilitação* [>. Na expressão $B_1 [> B_2$ a execução de B_1 pode ser interrompida pela ocorrência do evento inicial de B_2, o qual passa a ser executado no lugar de B_1. Caso B_1 termine com sucesso antes da ocorrência do evento inicial de B_2, este último deixará de existir (transforma-se no processo *stop*).

2.1.3 Tipos de Dados

Em LOTOS valores e estruturas de dados são descritos através de ACT ONE, sendo que nessa linguagem algébrica não há tipos de dados predefinidos. Os tipos mais comuns, incluindo-se os básicos (e.g., inteiros, booleanos), devem ser especificados ou importados de uma *biblioteca padrão*, que geralmente é construída paulatinamente pelo projetista na medida em que este vai se especializando na linguagem.

A definição de um tipo (*type*) é realizada especificando-se um conjunto de portadores de dados (*sorts*), um conjunto de operações (*opns*) e um conjunto de equações (*eqns*). As operações podem envolver *0* (operação *nulária*) ou mais *sorts* (e.g., operações *unárias*, *binárias*), sendo que no primeiro caso definem os termos que são as constantes do tipo enquanto que no segundo permitem gerar novos termos a partir dos já existentes. As equações definem a semântica do tipo, ou seja, como as operações se relacionam. Por exemplo, o tipo booleano pode ser especificado em ACT ONE da seguinte forma:

```
type Boolean is
  sorts Bool
  opns
    true, false:        → Bool
    not:          Bool → Bool
  eqns
    ofsort Bool
      not (true)  = false;
      not (false) = true;
endtype (* Boolean *)
```

Nessa especificação o tipo *Boolean* utiliza os portadores de dados *Bool*. As operações nulárias *true* e *false* definem que estas são as constantes de *Bool*, enquanto que a operação unária *not* permite gerar novos termos de *Bool* (e.g., *not(true)*, *not(false)*, *not(not(true))*). As duas equações definem a semântica da operação *not*, em relação às constantes *true* e *false*, permitindo que os termos gerados a partir dessa operação possam ser avaliados (e.g., *not(not(true)) = not(false) = true*).

Uma vez definido um tipo este pode ser estendido com novos *sorts* e/ou novas operações e/ou novas equações. Tipos também podem ser combinados para formar tipos

mais complexos, os quais passam a incluir os *sorts*, as operações e as equações dos tipos formadores. É possível ainda renomear um tipo, o que muda a sua sintaxe mas preserva a sua semântica. Tipos podem ser apenas parcialmente especificados (parametrização), deixando-se lacunas para o posterior preenchimento das mesmas (atualização), permitindo-se assim a reutilização desses tipos.

2.1.4 LOTOS Completo

LOTOS Completo (ou simplesmente LOTOS) é LOTOS Básico com ACT ONE, o que permite a definição e passagem de valores entre processos.

Em LOTOS a comunicação entre processos baseia-se na possibilidade destes oferecerem/aceitarem valores através de suas portas. A notação *a!E* significa que o processo está preparado para ofertar a expressão de valor *E* na porta *a*. A notação *a?x:t* significa que o processo está preparado para aceitar um valor do tipo *t* na porta *a*, o qual será assumido pela variável *x* que é do tipo *t*. Essa comunicação pode envolver mais de dois processos (*multi-way rendez-vous*) e pode ser estendida para múltiplas ofertas/aceitações numa mesma porta. Por exemplo:

```
process P [t, u, v]: noexit :=
      t ? x : nat; (u ! x; stop [] v ! x; stop)
endproc
```

O processo P pode receber um valor do tipo *nat* na porta *t*. Se ocorrer essa comunicação, esse valor será oferecido nas portas *u* e *v*. Essa escolha indeterminística será definida pelo ambiente, ou seja, quando um outro processo, que também tenha uma porta *u* ou *v*, estiver apto a aceitar esse valor.

Comportamentos condicionais são expressos em LOTOS através de guardas e de predicados de seleção, os quais definem condições a serem satisfeitas antes ou no momento da ocorrência da interação respectivamente. Por exemplo:

```
process P [t, u, v]: noexit :=
  t ? x : Bool;
  (
     [ x ] -> u ! x; stop          (* x = true ? *)
    []
     [ not (x) ] -> v ! x; stop   (* x = false ? *)
  )
endproc
```

```
process P[t, u]: noexit :=
  t ? x : nat [x < 10]; u ! x; stop
endproc
```

No primeiro processo a porta *t* pode receber um valor booleano que será atribuído à variável *x*. Se *x* for *true* o guarda *[x]* também o será, a primeira expressão de comportamento será escolhida e o valor *true* será oferecido na porta *u*. Se *x* for *false* o guarda *[not(x)]* será *true*, a segunda expressão de comportamento será escolhida e o valor *false* será oferecido na porta *v*.

No segundo processo o predicado de seleção *[x < 10]* impõe que, no momento da comunicação, o valor natural a ser recebido na porta *t* e a ser atribuído a variável *x* seja inferior a *10*. Se esta condição estiver sendo satisfeita a comunicação ocorre e o valor atribuído a *x* passará ser oferecido na porta *u*.

Além das portas formais uma declaração de processo pode conter também parâmetros formais. Um parâmetro declarado localmente a um processo pode ser associado à expressão de valor complexa, a ser recebida de um outro processo, aumentando assim a sua legibilidade na medida que essa expressão é substituída por um único identificador.

Em LOTOS uma lista de valores pode ser associada à terminação com sucesso (*exit*) de uma expressão de comportamento ou processo, sendo que a lista dos *sorts* desses valores é chamada de *funcionalidade da terminação*. Por exemplo, um processo que termine com sucesso pode passar esses valores a um outro processo parametrizado composto seqüencialmente (>>), desde que haja um casamento da funcionalidade da terminação do primeiro com os sorts dos parâmetros do segundo.

LOTOS possui construtores que permitem criar especificações mais compactas (e.g., escolha generalizada entre portas e valores) ou representar um número infinito de alternativas. Por exemplo, a expressão *(B[a1, h1] [] B[a2,h1] [] B[a3,h1])*, formada por três expressões de comportamento semelhantes, cuja única diferença é a primeira porta, pode ser abreviada para *(**choice** g1 **in** [a1, a2, a3] [] B[g1,h1])*.

Outra notação que permite especificações mais compactas é o construtor paralelo generalizado. Por exemplo, a expressão de comportamento *(B[a1, h1]* || *B[a2, h1]* || *B[a3,h1])* pode ser abreviada para *(**par** g1 **in** [a1, a2, a3]* || *B[g1, h1]).* Esse mesmo tipo de abreviação também pode ser empregado no entrelaçamento puro |||.

2.1.5 Especificação de Sistemas

Embora uma especificação em LOTOS seja um processo que representa um sistema como um todo, há algumas pequenas diferenças sintáticas nas descrições desses dois elementos da linguagem. A estrutura geral de uma especificação LOTOS é

```
specification S [a, c, d]: noexit :=
        type TipoQualquer is        (* definição dos tipos globais *)
                sorts SortQualquer
                opn OperQualquer : SortQualquer -> SortQualquer
        endtype   (*TipoQualquer *)
        behaviour                   (* definição do comportamento abstrato *)
                hide b in
                        P [a, b, c] |[b]| Q [b, d]
                where
                process P [t, u, v] ...   (* definição dos processos *)
                process Q [x, y] ...
endspec (* S *)
```

2.2 E-LOTOS

A experiência adquirida com LOTOS ao longo dos últimos quinze anos, demonstrou a sua adequação para a especificação e verificação de sistemas distribuídos e protocolos, sobretudo devido à definição matemática precisa dessa linguagem e a suas construções, que permitem descrever aspectos importantes desses sistemas, tais como indeterminismo, concorrência, sincronização e comunicação.

Entretanto essa experiência demonstrou também que LOTOS possui algumas limitações quanto a sua capacidade de expressão e estruturação, dificultando a descrição de determinados tipos de sistemas distribuídos (e.g., sistemas de tempo real). Para suprir

essas limitações foram propostas extensões para LOTOS, culminando na criação e padronização de uma nova TDF denominada *Enhancements to LOTOS* (E-LOTOS) (ISO, 2001; VERDEJOO, 2000). As extensões mais importantes foram:

- introdução de uma noção de tempo quantitativo, tornando possível definir o tempo no qual as ações e comportamentos poderão ocorrer;
- introdução de tipos de dados predefinidos;
- possibilidade de modularizar uma especificação através da definição de módulos e interfaces, sendo que um módulo pode possuir definições de tipos, funções e processos;
- tipificação de portas;
- definição de novos operadores.

E-LOTOS é uma linguagem formada por dois níveis. O nível inferior, chamado de Linguagem Base, é a parte usada para descrever o comportamento dos processos. O nível superior, conhecida como Linguagem Modular, é utilizada para modularizar especificações.

2.2.1 A Linguagem Base

Além de novos operadores, as principais diferenças de E-LOTOS com relação LOTOS estão na sintaxe de representação de comportamentos e nos tipos de dados.

2.2.1.1 Declarações

Na linguagem base uma especificação é descrita como uma seqüência de declarações de tipos de dados, funções e processos.

Declaração de Tipos

Em E-LOTOS existem estruturas de dados, tais como registros (*records*) e uniões (*unions*), e tipos primitivos, tais como *bool*, *nat*, *float*, *int*, *string* e *char*, que são predefinidos e que facilitam a declaração de novos identificadores de tipos. Por exemplo, a definição de um registro, que armazene as partes real e imaginária de um número complexo, pode ser realizada em E-LOTOS da seguinte forma:

```
type complexo is
    (real => float, imag => float)
endtype
```

Há também a possibilidade de especificação de um supertipo para registros através do operador *etc*. Por exemplo, o registro *nome => string, etc* é um supertipo para qualquer registro que possua pelo menos o campo *nome*, tal como *nome => string, idade => int*.

Além dos tipos primitivos apresentados, há o tipo vazio *none* que não representa valor e serve para indicar a funcionalidade de processos que nunca terminam. Há também o tipo universal *any*, que é um supertipo associado às portas que podem comunicar qualquer tipo de dado.

Além dos registros, outra forma para definir tipos é a união |, que descreve todos os construtores envolvidos. No exemplo,

```
type pdu is
    send(packet, bit) | ack(bit)
endtype
```

o tipo *pdu* é uma união dos construtores *send* e *ack*, com *packet* e *bit* previamente definidos.

Declaração e Iniciação de Funções

As funções em E-LOTOS podem ser apenas utilizadas com expressões de dados e nunca representam comportamentos. Sua definição formal é realizada através de um identificador de função, de uma lista de parâmetros, do tipo do valor retornado, de uma lista de exceções e das expressões que irão compor o seu corpo. No exemplo

```
function conjugado (c: complexo):complexo is
        (real => c.real, imag => -c.imag)
endfun
```

a função *conjugado* recebe como parâmetro de entrada um dado do tipo *complexo* e retorna um outro dado complexo com a parte imaginária invertida. A chamada desta função poderia ser

```
?c1 := conjugado(c2)
```

com c1 e c2 do tipo *complexo*, como definido na função.

Declaração e Iniciação de Processos

A declaração de processos, similarmente a de funções, possui um identificador de processo, uma lista de parâmetros, um tipo de resultado e uma lista de exceções. No entanto processos podem representar comportamentos e podem se comunicar através de portas. No exemplo

```
process Register [in:data, out:data]:exit (none) is
    var x:data in
        loop
            in?x; out!x
        endloop
    endvar
endproc
```

o processo *Register* representa um registrador com capacidade de armazenar um dado do tipo *data*, aceito através da porta *in* e oferecido na porta *out*. Não há nenhum retorno de resultado (*none*) e o processo nunca termina. A iniciação desse processo poderia ser realizada da seguinte forma:

```
Register [inGate1, inGate2]()
```

Comparando com a declaração de processos em LOTOS, as principais diferenças estão na tipificação das portas, na descrição da funcionalidade do processo e na representação de exceções.

2.2.1.2 Expressões de Comportamento

Em E-LOTOS um sistema também é especificado através de seu comportamento observável, descrevendo-se a ordem temporal em que os eventos desse sistema podem ocorrer. A maior diferença para LOTOS é que E-LOTOS possui um número maior de operadores, os quais serão apresentados a seguir.

Ações

Um sistema concorrente é especificado como um conjunto de processos comunicantes que interagem por meio de ações. As ações observáveis são aquelas que representam sincronização ou comunicação entre processos através de portas e são especificadas da seguinte forma:

G **[**P1**]** **[**@P2**]** **[**[E]**]** **[start** (N)**]**

A ocorrência de uma ação associada à porta G implica no recebimento e/ou encaixe de valores em P1 e P2, além da avaliação da expressão E como verdadeira após N unidades de tempo decorridas desde a habilitação da ação. Os componentes representados entre **[]** são opcionais, sendo que a sincronização mais simples é aquela onde somente a porta G é indicada. Por exemplo, o comportamento

inP (?x : int) @?t [t < 5]

especifica uma ação recebe um inteiro na porta *inP* e atribui esse valor à variável *x*, desde que menos de 5 unidades de tempo tenham decorrido a partir do momento em que a comunicação tornou-se possível em *inP*. A construção **[**@P2**]** permite representar comunicação sensível ao tempo, o que não é possível em LOTOS.

Similarmente a LOTOS também são definidos o evento interno *i* e o pseudo-evento δ para terminação com sucesso. E-LOTOS adiciona aos mesmos a ação *delay* (ε), que representa a passagem do tempo, e uma ação que representa exceção.

Atraso (delay)

O operador de atraso *wait(E)* introduz um intervalo de tempo, que é definido através da expressão E ou indeterministicamente. Por exemplo, a expressão

wait (3); **i**; **exit**

indica que uma ação interna ao sistema ocorre com um atraso de três unidades de tempo após sua habilitação. Já a expressão

wait (**any** time); **i**; **exit**

indica que a ocorrência dessa ação interna, após a sua habilitação, é indeterminística.

Sinalização e Tratamento de Exceções

Exceções são importantes para tratar erros e outros comportamentos indesejáveis de um sistema passíveis de ocorrer. E-LOTOS possui o operador *trap* para manipular exceções e as instruções *raise* e *signal* para sinalizá-las.

A instrução *signal* provoca uma exceção mas continua a execução do comportamento em questão. Para interromper totalmente a execução de comportamentos ao ser disparada uma exceção, deve-se utilizar a instrução *raise*. Para tratar a exceção capturada por *signal* ou *raise*, é utilizado o operador *trap*. No exemplo

```
process Register [in:data, out:data] raises [ Error ] is
   var x:data in
         loop
                  in?x; out!x
         endloop
         [> (wait (50); i; signal Error)
   endvar
endproc
```

```
    ...
trap
   exception Error  is anError endxn
  in
    Register [...] () [Error]
endtrap
```

o processo *Register* foi modificado para sinalizar um erro (*signal Error*) numa situação de falha que, neste caso, ocorre quando o registrador não devolve a entrada (*out!x*) dentro de um intervalo de 50 unidades de tempo. O operador *trap* captura o erro e permite a outro processo que utiliza o registrador ser alertado através da porta *anError*.

Inação e Bloqueio de tempo

A representação de comportamento indesejável de um processo (e.g., impasse) é realizada através dos processos básicos *stop* e *block*. Em E-LOTOS *stop* também representa um processo inativo, mas diferentemente de LOTOS o pseudo-evento ε, que permite a passagem do tempo, é oferecida. Com o processo *block* a passagem do tempo torna-se proibida.

Terminação

Semelhantemente a LOTOS a terminação com sucesso de um comportamento é expressa através do processo *exit [(E)]* onde a expressão opcional *E* representa um argumento a ser passado à outra expressão de comportamento. Para passar um valor não-determinado de um certo tipo utiliza-se *exit (any <expressão de tipo>)*.

Operador de Desabilitação

O operador *[>* é exatamente o mesmo de LOTOS, sendo que B_1 *[>* B_2 expressa a possibilidade de o comportamento B_1 ser desabilitado permanentemente pelo comportamento B_2.

Operador de Suspensão/Retomada

O operador *[X>* é uma extensão do operador de desabilitação e permite o prosseguimento da execução do comportamento interrompido. Em B_1 *[X>* B_2 se B_1 é interrompido por B_2, então B_1 é suspenso até que B_2 gere uma interrupção *X*, a qual permitirá a retomada da execução de B_1.

Composição Seqüencial

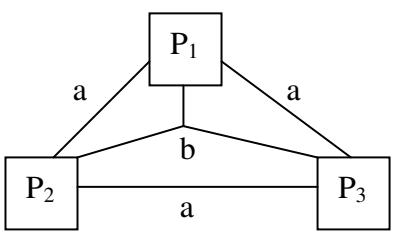
Em E-LOTOS a composição seqüencial *;* engloba o prefixo de ação *;* e o operador de habilitação >> de LOTOS, já que permite a composição seqüencial tanto de ações quanto de comportamentos. Entretanto, na composição seqüencial de duas expressões de comportamento não há geração do pseudo-evento δ para a habilitação da segunda expressão.

Composição Paralela

Além das três formas de composição paralela de LOTOS, E-LOTOS possui um *operador paralelo geral* que expressa diretamente redes de processos.

Como exemplo, vamos considerar o conjunto de *m* processos P_1, P_2, ..., P_m, onde se deseja obter um esquema de sincronização tal que a execução de uma ação numa determinada porta somente ocorra com a colaboração de uma coleção qualquer de *n* processos do conjunto. Quando $n = 2$, um processo qualquer P_i pode se comunicar através de uma porta com qualquer outro processo P_j com $i \neq j$ (sincronização de processos aos pares). Utilizando o novo operador, essa sincronização pode ser representada como no exemplo:

par a#2 **in**
  [a, b] → P_1
 || [a, b] → P_2
 || [a, b] → P_3
endpar



Neste exemplo, os processos P_1, P_2 e P_3 possuem as portas *a* e *b*. O operador **par** a#2 indica que a porta *a* deve ser sincronizada aos pares (#2) entre os processos. Como a porta *b* não consta como argumento do operador mas está presente em todas as listas de processos, sua sincronização ocorre entre todos os processos como mostra o esquema de sincronização acima.

A sintaxe do operador paralelo geral é dada por:

par $g_1\#n_1$, ... ,$g_p\#n_p$ **in**
    $[T_1]$ -> B_1
  | | ...
  | | $[T_m]$ -> B_m
endpar

onde ($g_1\#n_1$, ... ,$g_p\#n_p$) é uma lista de graus, tal que uma porta g_i está associada ao grau n_i , com $1 \leq i \leq p$ e $p \geq 0$, e T_i´s são listas de portas de sincronização $T_i = G_{i1}, ..., G_{ir}$. associadas aos respectivos comportamentos B_i, com $1 \leq i \leq m$.

O significado intuitivo desse operador é o seguinte: se um comportamento B_i pode realizar uma ação pela porta *G* e esta porta pertence à sua lista de sincronização T_i, então essa ação deve ser sincronizada com as ações dos demais comportamentos da seguinte forma:

- se *G* pertence à lista de graus com um grau *n*, então B_i se sincronizará em *G* com outros *n-1* quaisquer comportamentos que possuam *G* em sua lista de sincronização;
- se *G* não pertence à lista de graus, então B_i se sincronizará com todos os outros comportamentos que possuam *G* em sua lista de sincronização.

Por outro lado, caso *G* não pertença à lista T_i de B_i, não haverá sincronização de B_i por essa porta com os outros comportamentos.

Como LOTOS, E-LOTOS dispõe também de um *operador de paralelismo sobre valores*, que é utilizado para realizar diversas instanciações paralelas de um mesmo comportamento, que dependente de uma variável, de tal maneira que cada instância assuma um valor diferente para essa variável. A sua sintaxe é a seguinte:

par P **in** N ||| B **endpar**

onde *P* é um padrão que representa uma variável ou expressão, *N* é uma lista de valores que podem se assumidos por *P*, e *B* é o comportamento dependente das variáveis assumidas por *P*.

Renomeação

Em E-LOTOS é possível renomear ações observáveis e exceções, podendo-se alterar a estrutura de um evento que ocorre através de uma porta. No exemplo

```
rename
    gate inP (( x => ?a, y => ?b )): recordXY is inP (! ( x => a ))
 in
    B
endren
```

o processo *B* envia, através da porta *InP*, o evento do tipo registro *recordXY* composto dos campos *x* e *y*, os quais são de um tipo qualquer. Para que *B* possa se comunicar com outro processo, que aceita pela sua porta apenas um dos campos, a porta *inP* é renomeada para passar apenas o valor associado ao campo *x* do registro.

Operadores Condicionais

Assim como nas linguagens de programação, E-LOTOS possui o operador condicional

```
if E then B
  (elsif E then B)*
  [ else B]
endif
```

onde *E* deve ser uma expressão booleana e *B* um comportamento associado.

Para a avaliação de uma expressão de dados sobre um conjunto de possibilidades, E-LOTOS possui o operador condicional

case E **is**
(E -> B)*
endcase

onde *E* é uma expressão de dados e B um comportamento.

Características Imperativas

Para facilitar a especificação de sistemas E-LOTOS possui diversas características imperativas, dentre as quais destacam-se:

- A atribuição *P* **:=** *E* onde *P* é um conjunto de parâmetro de atribuição e *E* é uma expressão de dados. É possível realizar a atribuição de um valor aleatório a *P* através do operador *any*;
- A declaração local de variáveis *var*;
- O operador iterativo *loop*, que representa um laço infinito que somente pode ser interrompido por uma instrução *break* ou por uma interrupção. Também possui o laço *for* e a iteração condicional *while*.

2.2.2 Linguagem Modular

Em LOTOS somente tipos e operações podem ser encapsulados. Já em E-LOTOS é possível também a um módulo encapsular funções e processos, além de controlar quais objetos serão exportados pelo módulo (se tornarão visíveis ao ambiente) e quais ficarão ocultos.

Para facilitar a modularização, houve uma separação entre o conceito de módulo (definição dos objetos) e interface (visibilidade externa do módulo). Uma especificação em E-LOTOS modular é constituída de uma seqüência de declarações de interfaces, de módulos e de módulos genéricos.

2.2.2.1 Interfaces

A interface define a parte visível dos objetos (tipos, funções e processos) declarados dentro de um módulo. Esta descreve um tipo de módulo, que representa uma classe de módulos com a mesma visibilidade, e pode opcionalmente importar outras interfaces. A sua sintaxe é

interface int_id [import <identificadores de especificação de interface >] **is**
    i-body
endint

onde *int_id* é um identificador de interface e *i-body* é o seu corpo. Por exemplo, a interface de um módulo que implementa um registrador pode ser definida como

interface Register_Interface **is**
    type data
    process Register [in: data, out: data]
endint

sendo que qualquer módulo, que tenha pelo menos um tipo *data* e um processo *Register* com duas portas do tipo *data* e mesma funcionalidade, pode se associar a essa interface.

2.2.2.2 Módulos

Um módulo especifica a implementação de um conjunto de tipos, funções e procedimentos. Os objetos que um módulo exporta, os quais poderão ser importados por outros módulos, são controlados através de interfaces. A sua sintaxe é

module mod-id [: int-exp] [**import** <expressões de módulo>] **is**
    m-body
endmod

onde *mod-int* é um identificador do módulo e *int-exp* é uma expressão de interface que declara os objetos visíveis ao módulo. As expressões de módulo permitem a importação de outros módulos ou a instanciação de módulos genéricos. O corpo *m-body* do módulo

é composto por declarações da linguagem base, enriquecidas com funções que declaram valores constantes.

2.2.2.3 Módulos Genéricos

A generalização de módulos é um mecanismo utilizado na construção de especificações que permitam a reutilização de código. A sua sintaxe é

generic gen-id (mod-id :int-id) **is** m-body **endgen**

onde *gen-id* é um identificador de módulo genérico, *mod-id* é um identificador para um módulo formal com interface *int-id*. Esse módulo determina os objetos que poderão ser utilizados na definição do corpo *m-body* do módulo.

2.3 SDL

Specification and Description Language (SDL) é uma TDF padronizada pela ITU-T que permite representar sistemas tanto graficamente (*Graphical Representation - GR*) quanto textualmente (*Phrase Representation – PR*), aceitando ainda a introdução de texto informal na descrição.

O primeiro padrão SDL surgiu em 1976 como uma linguagem gráfica básica utilizada para a descrição de centrais de comutação telefônica. Em 1980, o padrão incorporou a definição de semântica de processos e, em 1984, os conceitos de estruturas de dados. O aperfeiçoamento de SDL culminou com o surgimento, em 1988, do SDL-88, quando foi estabelecida sua definição formal.

Desde então, SDL passou por diversos melhoramentos, levando ao surgimento do SDL-92 e do SDL-96, onde foram introduzidos à linguagem conceitos de orientação a objetos. Mais recentemente surgiu SDL-2000, que introduziu alterações pouco significativas ao SDL-96. Esta seção apresenta uma visão geral dos principais conceitos de SDL, sendo que tutoriais relativos a essa linguagem podem ser encontrados em (TURNER, 1993; FAERGEMAND; OLSEN, 1992).

2.3.1 Modelagem de Sistemas em SDL

Em SDL a especificação de um sistema descreve um conjunto de *blocos* conectados por meio de *canais* de comunicação. O bloco é o principal conceito de estruturação e representa um subsistema do sistema principal. Os canais permitem a interação entre blocos e com o ambiente. A Figura 4 demonstra graficamente um sistema exemplo estruturado em blocos e a sua sintaxe correspondente.

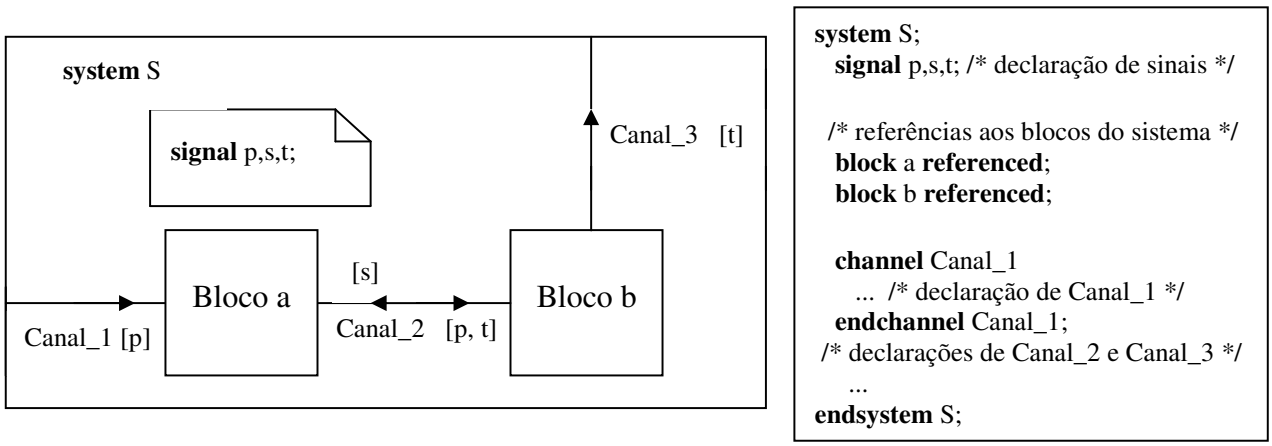
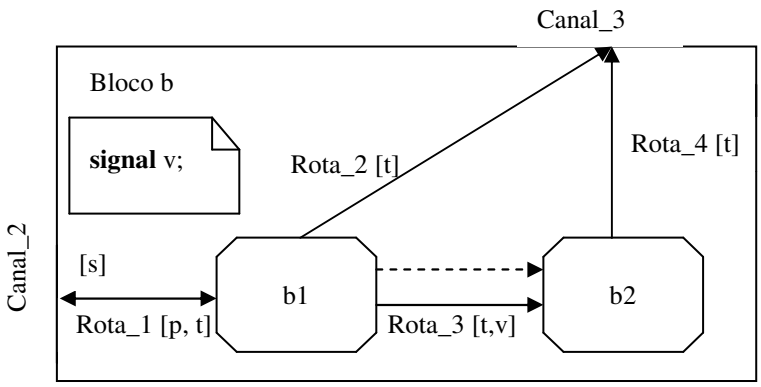


Figura 4: Representação de um sistema em SDL

Um sistema é composto de uma parte de declarações, onde se encontram as referências de blocos e as declarações de canais e sinais, e uma parte onde os blocos são especificados. Na Figura 4 o sistema *S* está estruturado em dois blocos (referenciados como *Bloco a* e *Bloco b*) conectados através de *canais*, indicados como segmentos de reta com uma ou duas setas em seu interior. Um canal pode ser uni ou bidirecional (e.g., *Canal_1* e *Canal_2* respectivamente) e são caracterizados pela *lista de sinais* que podem carregar. Os sinais são representados entre colchetes e acompanham o nome do canal na representação gráfica. Se um canal é bidirecional deve-se indicar quais são os sinais transportados em ambas as direções. A palavra-chave *signal*, presente na parte de declaração do sistema, indica o conjunto de sinais definidos para os canais naquele nível de abstração do sistema.

Os blocos são estruturados como um conjunto de processos comunicantes que operam em paralelo. Os processos são responsáveis pela definição do comportamento do sistema como um todo e são descritos como *Máquinas de Estados Finitas Estendidas (MEFEs)*. Analogamente a um sistema distribuído, um bloco representa um nó do sistema, enquanto que os processos representam as entidades que operam dentro do mesmo nó.

Assim como num subsistema, um bloco possui uma parte de declarações e uma parte de definição de processos. A Figura 5 apresenta o refinamento gráfico e textual do *Bloco B*. Este é composto por até dois processos interligados entre si e conectados (*connect*) com o ambiente através de rotas, representadas graficamente como segmentos de reta com uma ou duas setas em suas extremidades. Semelhantemente aos canais, os sinais transportados pelas rotas, que também podem ser uni ou bidirecionais, devem ser indicados. Rotas são conceitualmente diferentes de canais por não introduzirem atraso no transporte de sinais. Os canais, que podem representar uma rede de comunicação que interliga os nós do sistema distribuído, não entregam sinais instantaneamente, adicionando um atraso indeterminístico aos mesmos.




```
block b;
  signal v; /* sinal definido dentro do bloco*/

  process b1 referenced;
  process b2 referenced;

  signalroute Rota_1
    ... /* declaração de Rota_1 */
  ;
/* declarações de Rota_2, Rota_3 e Rota_4*/
    ...
  connect Canal_2 and Rota_1;
  connect Canal_3 and Rota_2;
  connect Canal_3 and Rota_4;
endblock b;
```

Figura 5: Refinamento do bloco b do sistema S

Processos podem criar dinamicamente processos filhos dentro do bloco ao qual pertencem. Na Figura 5 o processo *b1*, que age como um controlador para a criação e eliminação dinâmica de processos dentro do *Bloco b*, pode criar o processo *b2*, fato que é representado graficamente com uma linha pontilhada ligando os dois processos.

Um bloco herda todas as declarações realizadas em níveis de estruturação hierarquicamente superiores ao desse bloco. Assim sendo, na Figura 5 apenas o sinal *v* deve ser definido dentro do *Bloco b* através de *signal*, pois todos os demais sinais, representados dentro desse bloco, já foram definidos pelo sistema.

2.3.2 Comunicação

Em SDL a comunicação entre processos é realizada através de trocas de mensagens e, de uma maneira restrita, através de variáveis compartilhadas.

2.3.2.1 Comunicação por Sinais

Declaração de Sinais

Um processo pode receber apenas sinais que foram declarados nas listas de sinais das rotas e canais associados a este. A sintaxe de uma lista é

signal sinal_1 (*Parâmetros*), ..., sinal_n (*Parâmetros*);

sendo que os sinais declarados podem opcionalmente passar um conjunto de parâmetros entre processos. SDL dispõe de um conjunto de tipos predefinidos (e.g., booleano, inteiro, natural, real, string), que podem ser usados para tipificar esses parâmetros. Assim como em LOTOS, é possível especificar novos tipos em SDL na medida em que estes forem necessários.

Declaração de Canais e Rotas

Na declaração de um canal devem ser definidos quais blocos serão por este conectados. A mesma regra deve ser aplicada na declaração de rotas em relação aos processos. As sintaxes de canal e rota são

Declaração de Canal	*Declaração de Rota*
Channel < nome_do_canal > **from** < nome_do_bloco > **to** < nome_do_bloco > **with** < lista_de_sinais >; [**from** < nome_do_bloco > **to** < nome_do_bloco > **with** < lista_de_sinais >;] **endchannel** < nome_do_canal >;	**signalroute** < nome_da_rota > **from** < nome_do_proc > **to** < nome_do_proc > **with** < lista_de_sinais >; [**from** < nome_do_proc > **to** < nome_do_proc > **with** < lista_de_sinais >;]

sendo que a especificação entre colchetes é opcional e só deve ser empregada na definição de canais e rotas bidirecionais. De acordo com essa sintaxe, na Figura 4 o Canal_2, que interliga os blocos a e b, e o Canal_1 são definidos como

Canal_2 *(comunicação entre processos)*	*Canal_1* *(comunicação entre processo e ambiente)*
channel Canal_2 **from** a **to** b **with** p, t; **from** b **to** a **with** s; **endchannel** Canal_2	**channel** Canal_1 **from env to** a **with** p; **endchannel** Canal_2

Blocos e/ou processos também podem se comunicar com o ambiente, sendo que, nesse caso, utiliza-se a palavra-chave **env** no lugar do nome do bloco ou processo, tal como ocorre na definição do *Canal_1*.

Para que processos pertencentes a um bloco possam enviar sinais a processos localizados em outros blocos, suas rotas de comunicação devem ser conectadas convenientemente aos canais que interligam esses blocos. Por exemplo, na Figura 5 as *Rota_2* e *Rota_4* comunicam-se com o ambiente através de um ponto de conexão. O *Canal_3*, conectado a esse ponto, deve obrigatoriamente transportar o mesmo conjunto de sinais suportados pelas duas rotas.

A comunicação tanto em canais quanto em rotas é assíncrona. Há uma fila de entrada associada a cada processo, onde ficam armazenados os sinais até serem consumidos pelos mesmos.

Endereçamento de Processos

Identificadores *PId* são atribuídos automaticamente aos processos, na medida em que estes são criados estática ou dinamicamente pelo sistema. O acesso ao PId de um processo por outro processo pode ser realizado através de expressões pré-definidas, as quais retornam o identificador do processo atual (*self*), do último processo a enviar um sinal (*sender*) ao processo atual, do processo-filho mais recentemente criado (*offspring*) e do processo-pai (*parent*) do processo atual. Os valores de *self* e *parent* não se alteram durante o ciclo de vida do processo, enquanto que os valores de *sender* e *offspring* podem variar.

Dependendo da estruturação do sistema, um processo pode enviar um sinal sem a identificação do receptor. Por exemplo, na Figura 5 o único processo capaz de receber o sinal *v* enviado por *b1* é o processo *b2*. Entretanto, se um mesmo sinal pode ser encaminhado através de mais de um canal, o processo receptor deve ser explicitamente indicado pelo seu *PId* na construção <*sinal*> *to* <*ident_processo*>, onde *ident_processo* pode ser uma expressão de endereçamento pré-definida ou uma variável criada pelo Usuário.

Se o *PId* de um processo não estiver disponível, o que geralmente ocorre quando o processo receptor não pertence ao mesmo bloco do transmissor, pode ser utilizado o endereçamento implícito, onde é realizada apenas a indicação do canal que irá conduzir o sinal ao seu destino. A construção <*sinal*> *via* <*canal*> é utilizada nesse tipo de endereçamento. Por exemplo, o sinal *t* do processo *b1* pode ser enviado pelo *Canal_3*, o que seria indicado como *t via Canal_3*.

2.3.2.2 Compartilhamento de Variáveis

Uma alternativa ao uso de canais na comunicação entre processos é a utilização de variáveis compartilhadas. Valores de variáveis podem ser lidos ou modificados por processos de um mesmo bloco ou de blocos diferentes. Entretanto, esse artifício deve ser utilizado com restrições, para não criar dependências entre processos de blocos distintos.

2.3.3 Comportamento

Após a descrição da estrutura do sistema (referências a blocos e processos) e as interfaces associadas (listas de sinais, canais e rotas), o comportamento do sistema deve ser descrito através de processos.

Um processo é constituído de duas partes: um cabeçalho e um corpo. O cabeçalho define o nome do processo, o número de instâncias estáticas e dinâmicas que poderão existir e uma lista de parâmetros usados em sua iniciação. Um exemplo de cabeçalho é

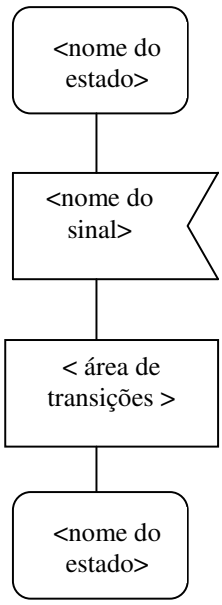
process proc (1, 5) **fpar** x Integer;

onde o processo *proc* terá uma instância criada estaticamente na iniciação do sistema e poderá ter no máximo 5 instâncias, significando que mais 4 instâncias podem ser criadas dinamicamente. O processo possui um parâmetro formal *x* do tipo inteiro. Caso o número de instâncias máximo não seja definido, é assumido que o processo poderá ter apenas uma ocorrência estática e infinitas ocorrências dinâmicas

Todo processo possui um comando *start*, que determina o ponto de início de sua execução, e que é ativado automaticamente logo após sua iniciação. Nesse ponto podem ser realizadas algumas iniciações (e.g., iniciação de variáveis) e se determina o estado inicial do processo. O encerramento do processo é realizado através do comando *stop*, que pode estar presente em diversos pontos do seu corpo.

O corpo de um processo é descrito como um conjunto de estados e transições que representam uma *MEFE*. Os estados determinam quais condições devem ser satisfeitas para que uma transição possa ser executada. As condições são formadas pelo estado vigente *state*, pelo sinal *input* esperado na porta de entrada do processo e, ocasionalmente, por uma expressão booleana. A declaração gráfica e textual dos estados e das transições segue a sintaxe apresentada na Figura 6.

Uma transição estará apta a ser executada se o estado do processo for igual ao seu estado vigente, se o sinal aguardado por esta estiver encabeçado na fila de sinais do processo e se a condição booleana, caso exista, esteja satisfeita. Após a execução dessa transição o processo pode assumir um novo estado.



state < nome do estado >

input < nome do sinal >;

< área de transições >

nextstate <nome do estado >

[**endstate** [< nome do estado >] ;]

Figura 6: Representação gráfica e textual de estados e transições

Uma transição pode ser iniciada apenas com a avaliação de uma expressão lógica, dispensando a necessidade de recepção de sinal. Transições dessa categoria são espontâneas, possuindo uma prioridade de execução inferior às demais transições.

Após a avaliação da condição de disparo da transição, o conjunto de ações associadas a esta é executado. As ações compreendem:

- *output* de sinais através de uma rota ou canal, conforme a forma de endereçamento;

- *tasks* que alteram o valor de variáveis que armazenam dados internos aos processos;

- criação (*create* <*processo*><*Lista_de_Parâmetros*>) dinâmica de instâncias de processos-filhos;

- *decision* que permite à transição seguir sua execução por dois ou mais caminhos diferentes conforme a avaliação de uma expressão condicional. A semântica de

decision é semelhante à de *case* das linguagens de programação, onde cada alternativa determina um estado que pode ser alcançado a partir da avaliação de uma expressão lógica;

- *call* que permite a chamada de procedimentos os quais representam a parte de um processo que é parametrizada;

- *set* e *reset* que permitem respectivamente ativar e reiniciar um temporizador predefinido dentro de uma transição. Este artifício é útil para a descrição de transições que possuem sua execução atrasada por um intervalo de tempo.

Todas essas ações possuem representação gráfica em SDL, que são inseridas entre os símbolos de estado inicial e final da transição. Também pode ser inserido um texto informal para representar, por exemplo, tarefas e expressões condicionais de decisões.

Se um sinal de entrada não é tratado por nenhuma transição, esse sinal é consumido (descartado) implicitamente pelo processo sem que se altere o seu estado. Entretanto é possível salvar um sinal, que não é usado no estado vigente, para ser utilizado posteriormente no próximo estado a ser alcançado pelo processo.

2.4 Estelle

Extended State Transition Language (Estelle) foi desenvolvida para a especificação de sistemas distribuídos, protocolos de comunicação e os padrões relativos ao modelo de referência *Open Systems Interconnection (OSI)*. O seu desenvolvimento teve início em 1981, sendo que esta se tornou um padrão ISO em 1989. Estelle faz parte de uma segunda geração de TDFs e possui alguns conceitos comuns a outras TDFs mais antigas, dentre as quais destaca-se SDL.

As próximas seções apresentam os principais conceitos de Estelle. Para maiores informações a respeito dessa TDF, sugere-se uma consulta aos tutoriais (LOPES DE SOUZA, 1989; BUDKOWSKI; DEMBINSKI, 1988; DIAZ et al., 1989).

2.4.1 Modelagem de Sistemas em Estelle

Estelle é baseada num modelo híbrido que associa *MEFEs* à linguagem de programação *Pascal* da ISO (ISO, 1983). Um sistema é descrito em Estelle através de uma estrutura hierárquica de *MEFEs* comunicantes, que podem ser executadas em paralelo e que se comunicam, de uma maneira geral, através de mensagens e, de uma maneira mais restrita, através de variáveis compartilhadas.

Estelle permite uma separação clara da descrição da interface de comunicação do comportamento interno dos diversos componentes de um sistema. Cada componente é representado por um módulo que, na verdade, é uma instância de um módulo definido dentro da especificação Estelle por uma definição de módulo.

Cada módulo possui uma interface associada à descrição do seu comportamento. A interface de um módulo é definida através de três conceitos:

- *pontos de interação* (portas): pontos de acesso ao módulo onde podem ser realizadas operações de entrada/saída;

- *canais*: a cada ponto de interação é associado um canal que define dois conjuntos de interações (um conjunto de mensagens que podem ser enviadas e outro conjunto de mensagens que podem ser recebidas);

- *interações*: interações são eventos abstratos (mensagens) que são trocadas entre os módulos de um sistema.

Informalmente, um módulo é representado graficamente como um retângulo junto com seus pontos de interação. A Figura 7 ilustra um sistema descrito em Estelle composto por três módulos. O módulo *S*, denominado de módulo sistema e representado na figura como um retângulo com linhas pontilhadas, engloba todos os demais módulos.

O sistema S está estruturado nos *Módulo_a*, *Módulo_b* e *Módulo*_c. Para que os módulos dessa hierarquia possam se comunicar por mensagens, *links* de comunicação (representados como segmentos de reta) são estabelecidos, ligando seus pontos de interação para formar os canais *Canal_1*, *Canal_2* e *Canal_3*.

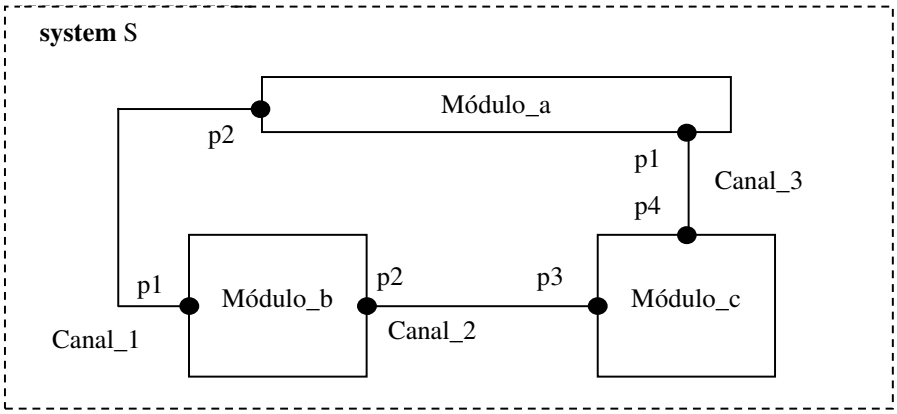


Figura 7: Representação de um sistema em Estelle

Um módulo em Estelle pode incluir a definição de submódulos denominados módulos filhos, o que permite refinar a especificação por eles representada. Os submódulos do mesmo módulo pai são chamados de módulos irmãos. A Figura 8 demonstra o refinamento do *Módulo_b* nos módulos *Módulo_b1* e *Módulo_b2*.

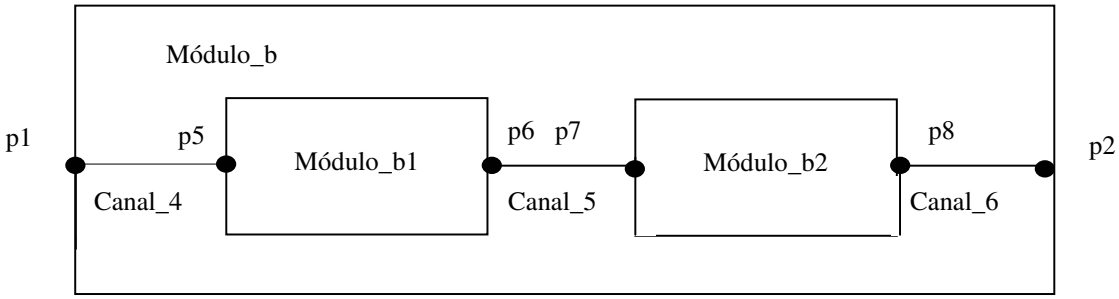


Figura 8: Refinamento do módulo b do sistema S

Os pontos de interação de um módulo podem ser externos (usados para a troca de mensagens com outros módulos) ou internos (usados para a troca de mensagens entre módulos irmãos ou entre os módulos filhos e o módulo pai);

Quando um ponto de interação externo de um módulo filho é ligado a um ponto de interação externo de seu módulo pai, dizemos que esses pontos estão vinculados (*attached*). Dois pontos de interação estão conectados (*connected*) se ambos forem pontos de interação de dois módulos irmãos ou se um deles for um ponto de interação externo de um módulo filho e o outro for um ponto de interação interno do módulo pai.

Na Figura 8, existe uma ligação do tipo *connect* entre os módulos *Módulo_b1* e *Módulo_b2* (p6 – p7) estabelecendo o canal interno *Canal_5*. Já a ligação entre os módulos *Módulo_b* e *Módulo_b1* (p1 – p5) e entre *Módulo_b* e *Módulo_b2* (p2 – p8) que estabelecem os canais *Canal_4* e *Canal_6* respectivamente, são do tipo *attach*.

2.4.2 Estrutura de Módulos

Um módulo em Estelle pode incluir a definição de submódulos como já demonstrado na Figura 7 em relação ao módulo sistema *S* e na Figura 8 em relação ao módulo *Módulo_ b*.

A maneira como as instâncias dos módulos se comportam umas em relação às outras é estritamente dependente do modo como eles estão aninhados e atribuídos. Todo módulo que possua transições (módulo ativo) deve possuir um tipo de *atributo* que determina a maneira como pode ser aninhado. As classes de atributos definidas em Estelle são:

- *systemprocess* e *systemactivity*: módulos com esses atributos definem subsistemas de comunicação separados dentro da especificação e não podem ser aninhados dentro de outros módulos ativos;

- *process* e *activity:* atributos para os outros módulos da hierarquia que são descendentes de subsistemas.

Módulos *systemprocess* ou *process* podem ser sub-estruturados em módulos *process* ou *activity*. Por outro lado, módulos *systemactivity* e *activity* só podem ser sub-estruturados em módulos *activity*. O número de subsistemas dentro de uma especificação é sempre fixo e cada um deles é a raiz de uma árvore de módulos.

Além da estruturação da especificação, deve ser considerado o tipo de paralelismo que os módulos do sistema devem ter ao fazer sua atribuição. Considerando-se o comportamento do sistema em relação à execução das transições que o definem, Estelle prevê:

- *execução paralela assíncrona:* nesse caso, cada subsistema do módulo de sistema elege e mantém um conjunto de transições em execução (suas e de seus descendentes) e esses conjuntos evoluem independentemente;

- *execução paralela síncrona:* a execução das transições dos submódulos de um módulo de sistema tem início simultaneamente e todas devem terminar antes que um novo conjunto possa ser eleito para execução;

- *não paralelismo:* o subsistema elege e mantém apenas uma única transição em execução por vez.

Módulos *systemprocess* e *systemactivity* são executados em paralelo de forma assíncrona. No interior de um módulo *systemprocess*, os módulos descendentes podem ser executados em paralelo, de forma síncrona. No interior de um módulo *systemactivity*, os módulos descendentes nunca podem ser executados em paralelo. Dentro de um subsistema, as transições dos módulos pai têm prioridade sobre as transições de seus módulos descendentes.

2.4.3 Comunicação

Em Estelle, o principal mecanismo de comunicação é a troca de mensagens através de canais de comunicação. Um canal é uma construção que permite desvincular a especificação das mensagens (interações) da especificação dos módulos, já que neste estas são descritas sem que seja necessária uma referência explícita aos módulos que serão conectados através do mesmo. Como são bidirecionais, os canais apresentam a sintaxe:

channel Ident_Canal (Papel1, Papel2)
by Papel1: Ident_Interação (Lista_de_Parâmetros);
.....
Ident_Interação (Lista_de_Parâmetros);
by Papel2: Ident_Interação (Lista_de_Parâmetros);
.....
Ident_Interação (Lista_de_Parâmetros);

onde *Papel1* e *Papel2* definem as funções (emissão e recepção) que os módulos, os quais serão conectados às extremidades desse canal via seus pontos de interação, deverão desempenhar. O módulo que desempenhar o *Papel1* poderá enviar as mensagens definidas em *Papel1* e deverá receber as definidas em *Papel2*, sendo que o contrário ocorre com o módulo que desempenhar o *Papel2*. Cada mensagem pode vir acompanhada de uma lista de parâmetros.

A comunicação em Estelle é assíncrona. Quando uma instância de módulo recebe uma interação, ela é colocada numa fila tipo *FIFO* (*First-In, First-Out*) de tamanho ilimitado associada ao ponto de interação correspondente.

Outra forma de comunicação, utilizada entre módulos pais e filhos, é através do compartilhamento de variáveis. Um módulo filho pode utilizar uma variável exportada, cujo conteúdo pode ser acessado por seu pai. Entretanto, a troca de mensagens também pode ser utilizada para esse fim.

2.4.4 Cabeçalho de Módulo

A definição do cabeçalho do módulo especifica um tipo módulo o qual identifica uma classe de módulos com a mesma visibilidade externa, ou seja, com os mesmos pontos de interação, variáveis exportadas e mesmo atributo de classe.

Sua sintaxe é apresentada abaixo:

```
module Ident_Cabeçalho Atributo (Lista_de_Parâmetros);
ip Ident_Porta: Ident_Canal (Papel) Tipo_Fila;
                    ...
   Ident_Porta: Ident_Canal (Papel) Tipo_Fila;
export Lista_Variáveis_Exportadas;
end;
```

O nome do cabeçalho, definido por *Ident_Cabeçalho*, é seguido opcionalmente por uma classe de *atributo* que determina como o módulo deverá ser aninhado dentro da estrutura de módulos e a maneira como será executado. A *Lista_de_Parâmetros*

representa os parâmetros que podem ser passados para o módulo no momento de sua criação pelo seu módulo pai.

No cabeçalho é definida ainda a vinculação entre as portas (pontos de interação) e os canais. Assim, através de uma porta *Ident_Porta* será possível trocar primitivas definidas em *Ident_Canal*, sendo que as primitivas definidas em *Papel* são aquelas que poderão ser enviadas. Dessa forma, é indicada a direção das interações. O atributo *Tipo_Fila* especifica se as mensagens recebidas por um ponto de interação serão colocadas em uma fila individual (*individual queue*) ou se mais de um ponto irá compartilhar uma fila comum (*commom queue*). Opcionalmente, pode ser determinada uma lista de variáveis exportadas.

Geralmente, é feita uma declaração de um corpo de módulo para cada cabeçalho definido. Entretanto mais de um corpo podem ser declarados para a mesma definição de cabeçalho. Assim, módulos com o mesmo cabeçalho podem ter comportamentos internos diferentes.

2.4.5 Corpo de Módulo

O corpo de um módulo define seu comportamento e , de maneira geral, compreende um conjunto de transições (que representam uma MEFE) e sua estrutura interna (submódulos e suas interconexões). Um módulo pode não ter transições, sendo, neste caso, denominado de inativo.

A sintaxe da definição do corpo de um módulo é a seguinte:

```
body Ident_Corpo for Ident_Cabeçalho;
   { definição do corpo }
end;
```

O *Ident_Cabeçalho* faz a associação do corpo do módulo com seu cabeçalho. Corpos de módulos podem ser definidos dentro da especificação ou declarados como externos, indicando que sua definição está presente em outra especificação. Nesse caso, utiliza-se a palavra-chave *external*, como exemplificado a seguir:

```
body Ident_Corpo for Ident_Cabeçalho;
external;
```

Tais módulos podem ser utilizados para representar entidades que interagem com o sistema mas que não fazem parte de sua definição.

A definição do corpo de um módulo é composta de três partes: *declarações*, *inicializações* e *transições*.

Parte de Declarações

A parte de declaração da definição de um corpo contém declarações Pascal (constantes, tipos, variáveis, procedimentos e funções) e declarações de objetos Estelle específicos, como canais, cabeçalhos e corpos de módulo, variáveis de módulo, estados e conjunto de estados, e pontos de interação internos.

É nessa parte que submódulos podem ser declarados, definindo-se assim uma estrutura hierárquica de módulos aninhados para a arquitetura do sistema;

Parte de Inicialização

A parte de inicialização especifica a iniciação da variável de controle (variável que determina onde cada nova instância do módulo em questão começará sua execução, ou seja, seu estado inicial) e de variáveis de estado adicionais (Pascal). É feita também a criação de instâncias estáticas de submódulos definidos dentro do corpo do módulo.

Na parte de inicialização, podem ser criados estaticamente *links* de comunicação entre os pontos de interação. Os pontos de interação dos módulos filhos podem ser conectados, e pode haver a vinculação entre os pontos de interação dos módulos filhos e os do pai.

Parte de Transições

Nas transições é descrito, textualmente, o comportamento representado na *MEFE* associada ao módulo. Cada transição é composta de duas partes: *condições* e *ações*. Dentro de um conjunto de condições as seguintes cláusulas definem a transição com *condição de disparo*:

- **from** (from A1,..., An, onde Ai (1 <= i <= n) é um estado de controle ou um identificador de conjunto de estados;

- **when** (when p.m, onde *p* é um identificador de ponto de interação e *m* é um identificador de interação);

- **provided** (provided B, onde B é uma expressão booleana);

- **priority** (priority n, onde n é uma constante não negativa);

- **delay** (delay(E1,E2), onde E1 e E2 são expressões inteiras não negativas)

Algumas cláusulas podem ser omitidas e só pode aparecer no máximo uma cláusula de cada categoria na condição de uma transição simples. O conjunto de condições de uma transição indica se aquela condição é disparável (ou pronta-para-disparo), ou seja, se suas ações podem ser executadas.

Uma cláusula *from* é satisfeita se a variável de controle de estado atual do módulo estiver entre aquelas listadas por *from*. A cláusula *when p.m* é satisfeita se a interação *m* estiver na cabeça da FIFO associada ao ponto de interação *p* no estado atual do módulo. A cláusula *provided B* é satisfeita se a expressão booleana *B* retornar *true* no estado atual do módulo. A cláusula *priority* possibilita associar uma prioridade à transição, dando-lhe uma maior ou menor preferência quanto à execução.

Algumas transições podem possuir uma cláusula *delay*. A intenção dessa cláusula é indicar que a execução da transição (caso esteja habilitada) deve ser atrasada. Dois tempos (E1 e E2) estão associados a um *delay*: o tempo mínimo que a transição deve ser atrasada e o tempo máximo que ela pode ser atrasada. Uma cláusula *delay* somente pode ser usada em transições espontâneas (sem a cláusula *when*).

Uma transição (sem a cláusula *delay*) é dita estar habilitada se cláusulas "*from*", "*when*" e "*provided*", presentes na condição da transição, estão satisfeitas no estado atual do módulo.

Uma transição é dita ser disparável num estado de módulo em um dado momento de tempo se:

- estiver habilitada e, caso tenha uma cláusula delay(E1,E2), essa transição deve permanecer habilitada pelo menos E1 unidades de tempo, e;

- tiver a prioridade mais alta entre as transições que estão satisfazendo (a) , onde "a prioridade mais alta" corresponde ao menor número não negativo.

A execução de uma transição pode compreender o conjunto de ações:

- alteração do estado do módulo, através da cláusula **to** (*to* A, onde A é um identificador de estado de controle);

- um bloco de ações, isto é, uma seqüência de declarações Pascal e Estelle (com extensões e restrições específicas de Estelle).

A cláusula *to* (por exemplo, *to* ABERTO) especifica o próximo estado de controle que será alcançado uma vez que a transição é disparada. Se omitido ou especificado pela palavra-chave *same*, o próximo estado é o mesmo que o estado corrente.

As extensões de Estelle ao Pascal são as declarações: *output*, *init*, *release*, *attach*, *detach*, *connect*, *disconnect*, *all*, *forone* e *exist*. Com essas extensões, torna-se possível dentro da parte de transições, criar e eliminar dinamicamente submódulos, estabelecer e desconectar *links* de comunicação, e enviar interações.

As restrições Pascal são:

- todas as funções declaradas não devem gerar efeitos colaterais;
- os ponteiros só podem ser utilizados dentro de construções Pascal;
- tipo *file* não pode ser utilizado;
- uso restrito da declaração *goto*;
- as declarações *read* e *write* não podem ser usadas;
- *conformant arrays* não podem ser utilizados.

2.4.6 Criação de Instâncias e de Links de Comunicação

Instâncias de módulos filhos e seus *links* de comunicação podem ser criados estaticamente na parte de inicializações do módulo pai, ou dinamicamente na parte de transições.

Criação Estática

Na parte de declaração de um módulo são especificadas as variáveis que irão fazer referência às instâncias de seus módulos filhos que serão criadas. Essas variáveis possuem o tipo do módulo que irão referenciar e possibilitam o estabelecimento de *links* de comunicação. A criação de componentes durante a inicialização define a arquitetura interna inicial do módulo.

Abaixo é demonstrado, de maneira sucinta, a criação estática dos módulos e *links* de comunicação do *Módulo_b* exemplificados na Figura 8.

```
module Modulo_b process;
  ip p1: Canal_1 ...
     p2: Canal_2 ...
  end;

  body Corpo_Modulo_b for Modulo_b
  begin
    { parte de declarações de Modulo_b }
    module Modulo_b1 process;
    ip p5: Canal_4 ...
       p6: Canal_5 ...
    end;

    body Corpo_Modulo_b1 for Modulo_b1
    begin   { definição do corpo }
    end;

   module Modulo_b2 process;
    ip p7: Canal_5 ...
       p8: Canal_6 ...
    end;

    body Corpo_Modulo_b2 for Modulo_b2
    begin   { definição do corpo }
    end;

    modvar
      B1: Modulo_b1;
      B2: Modulo_b2;

   { parte de inicialização de Modulo_b }
   initialize
   begin
     init B1 with Corpo_Modulo_b1;
     init B2 with Corpo_Modulo_b2;
     connect B1.p6 to B2.p7;
     attach p1 to B1.p5;
     attach p2 to B2.p8;
    end;

end; { Modulo_b }
```

Na parte de declarações do *Módulo_b* são declarados os módulos filhos *Módulo_b1* e *Módulo_b2* e as variáveis *B1* e *B2* que irão referenciar, respectivamente,

esses módulos após sua criação. Na parte de inicializações, as variáveis *B1* e *B2* serão associadas aos seus respectivos módulos através da declaração *init*.

As declarações *connect* e *attach* estabelecem os *links* entre os módulos irmãos e entre estes e seu módulo pai. A declaração *connect B1.p6 to B2.p7* conecta o ponto de interação p6 de *Módulo_b1* (referenciado pela variável de módulo B1) ao ponto p7 do *Módulo_b2* (referenciado pela variável de módulo B2). A vinculação dos pontos p5 do *Módulo_b1* e p8 do *Módulo_b2* ao seu módulo pai é feita pelas declarações *attach p1 to B1.p5* e *attach p2 to B2.p8*, respectivamente.

Criação Dinâmica

Instâncias de módulos podem ser criadas e destruídas dinamicamente, o que permite a um sistema descrito em Estelle alterar sua arquitetura à medida que evolui. Da mesma maneira que na criação estática, variáveis de módulo devem ser declaradas para que um módulo pai possa posteriormente criar e referenciar seus módulos filhos.

A criação dinâmica de módulos filhos ocorre, porém, na parte de transições do módulo pai, diferentemente do que acontece na criação estática. Para exemplificar esse processo, consideremos que os módulos filhos de *Módulo_b* sejam criados dinamicamente:

```
Module Modulo_b process;
  ip p1: Canal_1 ...
     p2: Canal_2 ...
  end;

  body Corpo_Modulo_b for Modulo_b
  begin
    { parte de declarações de Modulo_b }
    module Modulo_b1 process;
    ip p5: Canal_4 ...
       p6: Canal_5 ...
    end;

    body Corpo_Modulo_b1 for Modulo_b1
    begin   { definição do corpo }
    end;

   module Modulo_b2 process;
    ip p7: Canal_5 ...
       p8: Canal_6 ...
    end;

    body Corpo_Modulo_b2 for Modulo_b2
    begin   { definição do corpo }
    end;

  { parte de transições }
   ...

  trans
  when p1.conectar
  begin
     init B1 with Corpo_Modulo_b1;
     init B2 with Corpo_Modulo_b2;
     connect B1.p6 to B2.p7;
     attach p1 to B1.p5;
     attach p2 to B2.p8;
   end;

    ...

end; { Modulo_b }
```

Neste exemplo, o recebimento da primitiva *conectar* (*when p1.conectar*) pelo ponto de interação *p1* do *Módulo_b*, leva à criação dinâmica (indicada por *init*) dos módulos *Módulo_b1* e *Módulo_b2*, da conexão (*connect*) entre eles e da vinculação (*attach*) entre esses módulos e seu módulo pai. Esses dois novos módulos passariam, por exemplo, a gerenciar a conexão solicitada pelo ambiente ao seu módulo pai.

A eliminação de módulos e *links* de comunicação também pode ser feita na parte de transições através das declarações *release*, *terminate*, *disconnect* e *detach*.

Para eliminar seus módulos filhos, o *Módulo_b* poderia incluir a seguinte transição:

```
trans
 when p1.desconectar
 begin
    release B1;
    release B2;
 end;
```

No caso acima, como ocorreu com a criação, a eliminação ocorre após o recebimento de uma interação pelo ponto *p1*. A declaração *release* desfaz todas as conexões (*connect*) e vinculações (*attach*) antes de eliminar os módulos referenciados por B1 e B2. Mensagens não consumidas contidas nas filas dos pontos de interação desses módulos são transferidas para as filas dos pontos de interação do *Módulo_b* ao qual estavam vinculados.

A declaração *terminate* executa a eliminação abrupta de módulos sem desconectar e desvincular pontos de interação. Mensagens presentes nas filas dos pontos do módulo eliminado são perdidas. Para se obter o mesmo efeito de *release* com *terminate*, deve-se empregar as declarações *disconnect* e *detach*, que explicitam a realização da desconexão e desvinculação dinâmica de pontos de interação.

Outra forma de sinalizar a eliminação do módulo filho, alternativamente ao recebimento de mensagens ao módulo pai para esse fim, é através de variáveis exportadas. Nesse caso, o evento que determina a eliminação do módulo é gerado por ele mesmo, e não pelo ambiente externo.

2.4.7 Semântica

A semântica de Estelle é operacional. Isto significa que uma relação-de-próximo-estado é definida sobre o conjunto de estados global do sistema que aqui são chamados de situações globais. A relação-de-próximo-estado especifica todas as possíveis situações que podem ser diretamente alcançadas a partir de uma dada situação. O comportamento do sistema como um todo é então caracterizado pelo conjunto de todas as situações globais que podem ser geradas (através de uma relação-de-próximo-estado) a partir de uma determinada situação inicial.

Como já foi visto, um sistema em Estelle é especificado como um conjunto de subsistemas, cujas instâncias de módulo dentro de cada subsistema podem executar suas transições de uma maneira paralela ou sem paralelismo, sendo que instâncias de módulos pertencentes a diferentes subsistemas podem executar suas transições de forma paralela assíncrona.

2.4.7.1 Situações Globais

Cada situação global de um sistema de transições é composta da informação corrente sobre:

- a estrutura hierárquica das instâncias de módulo do sistema especificado SP, a estrutura dos links estabelecidos entre seus pontos de interação e o estado local de cada instância de módulo. Toda essa informação é incluída em uma descrição instantânea global (Global Instantaneous Description – gid), escrito resumidamente como gid(SP);

- as transições que estão em "execução paralela síncrona" dentro de cada subsistema; o conjunto dessas transições para o i-ésimo subsistema é denotado por Ai (i=1,...,n, onde n é o número de subsistemas).

Cada situação global é denotada por:

$$sit = (gid(SP); A1,...,An)$$

Uma situação global é dita ser inicial se o "gid(SP)" é inicial e se todos os conjuntos Ai estiverem vazios. A "gid(SP)" é inicial se ela resulta da parte de inicialização da especificação SP.

2.4.7.2 Relação de Próxima Situação

Esta relação define as situações sucessivas de uma situação corrente arbitrária

(gid(SP); A1,...,Ai,...,An).

Ela é definida da seguinte forma:

Para todo i = 1,2,...,n,

- Se, na situação corrente, Ai estiver vazio (Ai = ∅), ou seja, não oferecer transições para execução, então a próxima situação será (gid(SP); A1,..,AS(gid(SP)/i),...,An) onde AS(gid(SP)/i) é o conjunto de transições selecionadas pelo i-ésimo subsistema para execução;

- Se, na situação corrente, Ai não estiver vazio, (Ai ≠ ∅), então a próxima situação será (t(gid(SP)); A1,...,Ai-{t},...,An) onde a nova gid(SP) resulta da transição *t* e *t* é removida do conjunto Ai.

A situação (1) indica que Ai, não tendo transições para executar, dá oportunidade aos componentes do i-ésimo subsistema de Ai oferecer transições. A situação (2) indica que Ai tem uma transição *t* e que ela será executada.

A execução de uma transição *t* de uma instância de módulo:

- pode alterar o estado local do módulo (como criar uma nova instância de um módulo filho e/ou um novo link de comunicação) ou gerar uma "saída" (enviar uma interação). Todas essas alterações são expressas por t(gid(SP));

- não pode influenciar a escolha das transições que já estiverem sido selecionadas por outros subsistemas ou a escolha de transições dentro do mesmo subsistema.

A seleção de transições do i-ésimo subsistema para serem executadas dentro de um passo de computação (ou seja, a escolha do conjunto AS(gid(SP)/i)) é regulamentada:

- pelo princípio de prioridade pai/filho, e;

- pelos atributos das instâncias.

A regra aplicada a uma instância de módulo dentro de um subsistema pode ser formulada da seguinte maneira:

- se uma instância de módulo tem uma transição disparável para oferecer, então esta será selecionada (prioridade pai/filho);

- caso contrário, dependendo do atributo do módulo ser "process" ("systemprocess") ou "activity" ("systemactivity"), respectivamente, todas ou uma (escolhida de forma indeterminística) de todas as transições disparáveis oferecidas pelas instâncias de seus módulos filhos, serão selecionadas.

O conjunto AS(gid(SP)/i) é gerado aplicando a regra acima recursivamente a partir do módulo raiz (sistema) do i-ésimo subsistema.

2.5 Comparação entre as TDFs

A Figura 9 representa uma tabela onde estão comparados os principais conceitos de projeto, responsáveis pela descrição da arquitetura e comportamento de um sistema, das TDFs estudadas nas seções anteriores. Os conceitos de projeto genéricos que definem uma arquitetura são os componentes que a constitui e os meios de interação que conectam esses componentes e permitem que trabalhem em conjunto para realizar o

comportamento global do sistema. Processos podem representar componentes em LOTOS, E-LOTOS e SDL, sendo que em Estelle essa representação é assumida pelo conceito de módulo.

Conceitos de Projetos de Sistemas	LOTOS	E-LOTOS	SDL	Estelle
Componentes	Processo	Processo	processo	Módulo
Meios de Interação	Portas	Portas	Canais, Rotas e pontos de conexão	Canais e pontos de interação
Comportamento	Álgebras de Processos	Álgebras de Processos	MEFE	MEFE
Paralelismo	Entrelaçamento de comportamentos	Entrelaçamento de comportamentos	Concorrência entre processos	Concorrência ou entrelaçamento entre módulos
Interações	Síncronas	Síncronas	Assíncronas	Assíncronas

Figura 9: Comparação entre TDFs

Os meios de comunicação podem ser síncronos, em que a interação só ocorre se os componentes envolvidos estão prontos para realizar a comunicação, ou assíncronos, em que as interações são armazenadas no caso do componente receptor não estiver pronto para receber interações no mesmo instante em que ocorrem. LOTOS e E-LOTOS descrevem comunicação síncrona através de portas, enquanto que SDL e Estelle adotam comunicação assíncrona para a descrição dos meios de comunicação, representados, em SDL, como canais, rotas e conexões, e, em Estelle, como canais e pontos de interação. As interações são definidas como síncronas ou assíncronas conforme o tipo do meio de comunicação onde trafegam.

Comportamentos são definidos no interior dos componentes. Sua descrição pode ser feita através de álgebras de processos, como ocorre em LOTOS e E-LOTOS, ou utilizar conceitos estruturais específicos para a descrição de MEFEs, como é o caso de SDL e Estelle. Um comportamento complexo pode ser dividido em um conjunto de

unidades de comportamento. As unidades de comportamento que representam uma MEFE, por exemplo, correspondem às condições e ações SDL ou às transições Estelle.

Os comportamentos definidos pelos componentes podem ser executados de maneira seqüencial ou paralela, sendo que, no paralelismo, pode haver concorrência ou entrelaçamento. Na concorrência, a execução de comportamentos distintos se sobrepõem no tempo, enquanto que no entrelaçamento, somente uma unidade de comportamento pode ser executada num determinado instante. Nos comportamentos especificados em LOTOS e E-LOTOS ocorre o entrelaçamento com sincronização através de portas. Processos SDL são concorrentes e os módulos Estelle podem ser concorrentes, se módulos sistema, ou apresentar entrelaçamento de comportamento, se sub-módulos processo de um sistema.

2.6 Ferramentas para TDFs

Vários tipos de ferramentas foram desenvolvidas para as TDFs apresentadas.

- *editores*: como todas essas TDFs possuem uma representação textual, editores de texto convencionais podem ser utilizados para escrever especificações. Visando facilitar o trabalho do especificador, foram concebidos *editores orientados à sintaxe*, que detectam erros e orientam on-line a produção de especificações, e *editores gráficos*, que representam especificações na forma de diagramas;

- *compiladores*: detectam erros léxicos, sintáticos e de semântica estática no código fonte de uma especificação. Na ausência de tais erros, esse código fonte é geralmente traduzido para um formato intermediário, que pode ser usado por outras ferramentas ou para gerar código em alguma linguagem de programação, auxiliando assim na implementação do sistema especificado;

- *verificadores e simuladores*: Um simulador executa simbolicamente uma especificação, permitindo a aplicação de seqüências de testes, as quais buscam detectar erros lógicos no projeto. Geralmente o Usuário tem como opção a execução interativa, onde a interação com o simulador pode ser feita a cada passo da

execução, ou automática, onde a execução apenas é interrompida ao alcançar um estado predeterminado. Um verificador ajuda a provar as propriedades de uma especificação analisando as propriedades a esta requeridas.

LOTOS

Diversas ferramentas estão disponíveis para LOTOS, dentre as quais destacam-se:

MiniLITE (MINILITE, 2004)

Lotos Integrated Tools Envirnment (*LITE*) é um conjunto de ferramentas projetado para computadores *Sun* com sistema operacional *Sun OS 4*. Dispõe de um ambiente (*Lite*) onde podem ser ativadas todas as demais ferramentas, que incluem um compilador e um verificador de semântica estática (integrantes do *Toolset for Product Realisation with LOTOS - Topo*), um gerador que converte a especificação para código *C* (*Colos*) e um avaliador simbólico (*Symbolic Interactive LOTOS Execution - Smile*). O avaliador é capaz de analisar partes isoladas ou toda a especificação, permitindo sua execução de forma interativa ou automática. Como resultado, podem ser gerados relatórios que procuram demonstrar se a especificação atende aos requisitos de sistema. O *MiniLite* também oferece ferramentas que facilitam a visualização e a interpretação desses relatórios.

Environment LOTOS de l'Université de Ottawa (*Eludo*) (XELUDO, 2004)

É um ambiente com ferramentas voltadas para a análise e a simulação de especificações *LOTOS*. A simulação pode ser conduzida de três modos diferentes: modo interativo (*Interactive System for LOTOS Applications - ISLA*), modo de geração de árvore simbólica (*Symbolic Expander of LOTOS Applications - SELA*), onde o usuário pode gerar uma árvore de todas as ações que podem ocorrer a partir de um determinado ponto e modo orientado a objetivo (*Goal-Oriented Execution - GOE*), onde o usuário solicita à ferramenta que encontre um caminho baseado num traço predefinido.

Eludo possui ainda ferramentas auxiliares que verificam a árvore gerada pelo *SELA* (*LOTOS Model Checker - LMC*) e que traduzem uma especificação *LOTOS* para *Prolog* (*LOTOS to Prolog Translation*), que é utilizado por outras ferramentas.

CAESAR /ALDEBARAN Development Package (*CADP*) (GARAVEL; LANG; MATEESCU, 2001)

É um conjunto de ferramentas voltado para a engenharia de protocolos. É constituído principalmente de dois compiladores, *CAESAR* e *CAESAR.ADT*, e do verificador *ALDEBARAN*. O *CAESAR* traduz a parte comportamental de uma especificação *LOTOS* para código *C* ou para um *Sistema de Transições Rotuladas* (*LTS – Labeled Transitions System*). O *CAESAR.ADT* traduz a parte de dados da especificação para bibliotecas de tipos e funções em *C*. O *ALDEBARAN* usa como entrada os *LTSs* gerados pelo compilador *CAESAR*.

E-LOTOS

Como a padronização de *E-LOTOS* é recente, novas ferramentas estão surgindo para dar suporte a essa TDF. Dentre estas destaca-se o compilador *Traian* (TRAIAN, 2003), que foi na realidade desenvolvido para uma variante de *E-LOTOS* denominada *LOTOS NT* (SIGHIRENU, 2003). Essa linguagem, segundo os autores do compilador, segue os principais conceitos de *E-LOTOS* e oferece também novas características para aumentar a eficiência da compilação e verificação de especificações.

Outra alternativa que tem sido empregada é a utilização de ferramentas que convertem uma especificação *E-LOTOS* para uma equivalente *LOTOS*, possibilitando assim a utilização de determinadas ferramentas *LOTOS* (e.g., as que integram o *CADP*).

SDL

Por se tratar da mais antiga TDF e ser utilizada sobretudo pela comunidade de telecomunicações, SDL possui a maior quantidade de ferramentas comerciais disponíveis. Algumas são apresentadas abaixo.

Sinderella SDL (SINDERELLA, 2004)

É uma ferramenta comercial de modelagem visual desenvolvida pela empresa dinamarquesa Cinderella para o desenvolvimento de sistemas de software, serviços e protocolos de comunicação e qualquer sistema baseado em troca de sinais ou mensagens. Possui uma notação gráfica orientada a objetos, que suporta as fases de análise, projeto, implementação e teste.

Essa ferramenta conta com um editor orientado à sintaxe, denominado *Cinderella Incremental SDL Analyser*, que pode corrigir automaticamente alguns tipos de erros durante a escrita da especificação. Seu simulador dispensa a pré-compilação da especificação, permitindo ainda realizar simulações parciais (isoladas) ou alterar a especificação durante a simulação.

SDL Integrated Tool Environment (*SITE*) (SITE, 2004)

É um conjunto de ferramentas formado principalmente por compiladores independentes capazes de traduzir uma especificação *SDL* para as linguagens *C++* e *Java*. É um ambiente de desenvolvimento aberto, o que possibilita a integração com outras ferramentas *SDL*. Os compiladores *SITE* podem, por exemplo, trabalhar em conjunto com o editor gráfico de *Cinderella*.

Para a análise de desempenho de sistemas escritos em *SDL* há o simulador *QUEST* (DIEFENBRUCH; HINTELMANN; MÜLLER-CLOSTERMANN, 1996), que adiciona à especificação dados sobre a arquitetura onde será executado o sistema especificado. As medidas de desempenho são exibidas graficamente e podem ser observadas em tempo real da simulação. O nome *QUEST* deriva do fato desse simulador ser baseado na linguagem *Queueing SDL* (*QSDL*), que estende SDL para

facilitar a modelagem de sistemas onde um conjunto de processos enfileirados aguardam para acessar um recurso compartilhado.

Telelogic TAU SDL Suite (TELELOGIC, 2004)

A Telelogic TAU desenvolveu um conjunto de ferramentas gráficas denominado *SDL Suite* capaz de exibir o comportamento e a estrutura de um sistema em diversos níveis de abstração. Com suporte para SDL-2000, as ferramentas incluem principalmente um editor SDL gráfico (*SDL Editor*) que integra diagramas *Unified Modeling Language* (*UML*) (UML, 2006) como se fossem parte integral de SDL, e um simulador de desempenho (*Performance Simulator*).

São fornecidos ainda compiladores capazes de traduzir uma especificação para código C ou C++ que pode ser adaptado para diversas arquiteturas. *SDL Suite* pode se integrar com outras ferramentas para realizar simulação de especificações. Seu ambiente gráfico facilita a condução da simulação e a análise dos resultados obtidos.

ObjectGeode (OBJECT, 2004)

O ambiente *ObjectGeode* é constituído de ferramentas gráficas sensíveis a contexto. Seu editor, por exemplo, é capaz de detectar erros sintáticos durante a escrita da especificação.

É constituído basicamente de um editor SDL, um editor MSC para a construção de gráficos de seqüência de mensagens (*Message Sequence Charts – MSC*), um simulador SDL interativo (*Model Debugger*), um gerador automático de testes (*Test Composer*) e um gerador de código C (*SDL C Code Generator*). Outras ferramentas, inclusive de outros fornecedores, podem ser integradas ao ambiente.

Safira SDL (SAFIRA, 2004)

Safira SDL é um ambiente integrado para implementação, validação e observação do funcionamento de sistemas. É composto basicamente por um editor gráfico *(Safire Editor)* que mostra uma visão hierárquica do sistema, e por um simulador *(Safire Campaigner)* que realiza a execução interativa do sistema especificado.

Ferramentas auxiliares ajudam na observação do comportamento do sistema em diferentes níveis de detalhe e na visualização dos resultados obtidos após uma simulação. Safira SDL possui ainda um analisador de protocolo capaz de capturar a troca de sinais entre os componentes de um sistema.

Estelle

Estelle Development Toolset (*EDT*) (EDT, 2000)

O primeiro ambiente integrado de ferramentas para *Estelle* foi o *Estelle Work Station* [39], desenvolvida pelo *European Strategic Programme for Research and Development of Information Technology (ESPRIT)* dentro do projeto *Software Environment for the Design of Open Distributed Systems (SEDOS) Estelle Demonstrator* (projeto No. 1265).

Baseado na experiência adquirida no projeto *SEDOS Estelle Demonstrator*, a *BULL S.A.* desenvolveu o *EDT*. Esse ambiente é composto de um conjunto de ferramentas formado principalmente pelo compilador *Ec (Estelle-to-C)* (BULL, 2000a) e pelo simulador *Edb (Estelle Debbuger)* (BULL, 2000b).

O *Ec* traduz uma especificação *Estelle* para o código *C* independentemente da plataforma de implementação. Este também gera um arquivo de representação intermediária (*IF – Intermediate Form*) que serve de entrada para o simulador *Edb*.

Edb é um simulador/depurador simbólico e interativo. É baseado diretamente no modelo semântico definido no padrão *ISO 9074* da TDF *Estelle*. *EDT* também possui a

ferramenta *Ug (Universal Generator)*, que auxilia na partição de especificações para a implementação de sistemas distribuídos.

Portable Estelle Translator/Distributed Implementation Generator (*PET/Dingo*) (SIJELMASSI; STRAUSSER, 1991a)

É um conjunto de ferramentas integradas para *Sun SPARC*, que traduzem uma especificação *Estelle* para código *C++*. O *PET* verifica a sintaxe de uma especificação e o *Dingo* produz o código, auxiliando também na implementação do sistema (SIJELMASSI; STRAUSSER, 1991b).

3 Abordagem Baseada em um Framework Conceitual e em Estelle

A abordagem para o desenvolvimento de sistemas de arquivos paralelos distribuídos, proposta nesta tese, estabelece como ponto de partida de projeto a elaboração de um *framework* conceitual, que descreve a arquitetura e o comportamento elementares compartilhados pelos sistemas que pertencem a essa categoria. Essa arquitetura é representada através de conceitos de projeto da *Unified Modeling Language* (UML) e constitui um modelo básico para a especificação e implementação de sistemas de arquivos mais elaborados.

Para a introdução de técnicas de descrição formal no ciclo de desenvolvimento de sistemas de arquivos paralelos distribuídos, a arquitetura UML do *framework* conceitual deve ser mapeada para conceitos de projeto de uma TDF, dando origem a uma especificação da arquitetura. A escolha da TDF mais apropriada, entre as apresentadas no Capítulo 2, para realizar essa especificação, advém da comparação da capacidade descritiva que cada TDF oferece para representar os conceitos UML.

Uma vez escolhida a TDF e realizado o mapeamento de conceitos, obtém-se uma arquitetura de um sistema básico, composta por um conjunto fundamental de processos, os quais devem estar presentes para que se possa caracterizar um sistema de arquivos paralelos distribuídos minimamente funcional. Isso implica que somente as operações mais elementares, compartilhadas pela maioria dos sistemas de arquivos, são consideradas nesse modelo.

Operações que explicitam a fragmentação dos arquivos, ao explorar de maneira mais otimizada o paralelismo inerente ao sistema, bem como mecanismos de controle de recursos compartilhados não são incluídos, uma vez que tais operações e mecanismos apresentam variações significativas nos sistemas de arquivos estudados, impossibilitando uma descrição generalizada dos mesmos que pudesse ser agregada ao *framework* conceitual.

Para a descrição formal de um sistema específico, que apresentam características próprias que não foram capturadas pela arquitetura básica, a especificação obtida a partir do *framework* conceitual deve ser complementada para satisfazer requisitos funcionais do sistema. Alterações podem ser operadas sobre a arquitetura (e.g., a definição de novos componentes) ou sobre comportamentos pré-definidos. Essa

complementação deve ser realizada sobre a especificação, sendo que as propriedades inerentes à TDF contribuem para garantir a consistência da descrição das novas características. O resultado é uma especificação que reflete o comportamento de todo o sistema, abstraindo apenas detalhes dependentes do ambiente de implementação. Nessa etapa é possível simular a especificação, na busca por erros de projeto e de descrição, antes da fase de implementação.

A implementação do sistema, a partir de sua especificação formal, pode ser obtida semi-automaticamente com o auxílio de ferramentas, que mapeiam os conceitos da TDF em conceitos de implementação de uma linguagem de programação. As funções, relacionadas diretamente ao ambiente onde o sistema será executado, devem ser codificadas manualmente e ligadas ao código gerado pelas ferramentas, complementando assim a especificação e finalizando o processo de implementação.

3.1 Framework Conceitual para um Sistema Básico

A elaboração de um *framework* conceitual é iniciada com a definição do domínio das aplicações. No Capítulo 1 foram estudadas as características comuns a diversos sistemas de arquivos paralelos distribuídos. Segundo essa generalização, um sistema básico deve ser constituído por um conjunto de processos Clientes, responsáveis pela intermediação do acesso dos Usuários ao sistema de arquivos, e por um conjunto de processos Servidores, responsáveis pelo armazenamento e recuperação dos dados no sistema local de arquivos onde estão em execução. A comunicação é realizada segundo o paradigma cliente/servidor, com os processos distribuídos por uma rede local.

Essa descrição permite a definição informal da arquitetura de um sistema de arquivos básico em um alto nível de abstração, como ilustrado na Figura 10.

Nesta arquitetura o processo Cliente é um processo do Usuário, o qual utiliza um conjunto disponível de primitivas para realizar operações sobre os arquivos distribuídos. O Cliente atua fragmentando e desfragmentando dados, acessando os Servidores e oferecendo ao Usuário uma operação transparente sobre os arquivos.

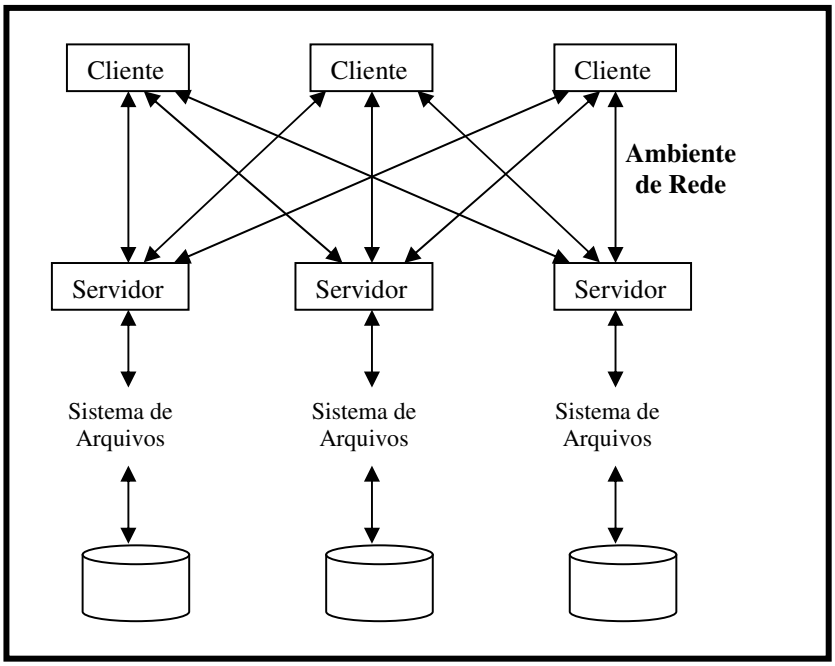


Figura 10: Arquitetura de um sistema de arquivos paralelos distribuídos

O Servidor é responsável pelo acesso aos fragmentos de arquivos que estão armazenados localmente. Cada computador, que possua um disco local compartilhado, deve ter uma cópia desse processo.

A arquitetura descentralizada foi considerada, para a elaboração do *framework* conceitual para o sistema básico, porque envolve o menor número possível de processos diferentes. Entretanto isso implica que os meta-dados de um arquivo distribuído, que incluem informações relativas à situação e ao ponteiro desse arquivo, aos Servidores onde este está distribuído, e ao tamanho da unidade de distribuição, são distribuídas entre os Servidores de dados, que passam também a assumir o papel de Meta-servidores. A maneira como os Clientes localizam e interagem com os Meta-servidores, para realizar uma determinada operação, pode apresentar variações entre os sistemas, motivo pelo qual essa questão não é tratada pelo *framework*. A solução mais adotada será ilustrada na descrição formal de um sistema de arquivos descentralizado, derivado diretamente do modelo básico.

3.1.1 Primitivas de Serviço

O *framework* conceitual identifica um conjunto comum de primitivas de serviço, que deve estar presente em qualquer sistema de arquivos distribuído. Diferentes sistemas oferecem primitivas especiais, as quais exploram de maneira otimizada o paralelismo, sendo mais adequadas quando o desempenho é prioridade sobre a transparência em relação à distribuição de dados. Porém é comum esses sistemas disponibilizarem primitivas de serviços que seguem a sintaxe Unix, mais familiar aos programadores. Essas primitivas possibilitam que programas, desenvolvidos originalmente para sistemas de arquivos locais, possam ser facilmente adaptados para serem executados sobre sistemas de arquivos paralelos, sem que o programador necessite conhecer detalhes específicos do funcionamento do novo sistema de arquivos.

Sete primitivas com a sintaxe Unix são definidas pelo *framework*, sendo que novas primitivas, visando a melhorar a exploração do acesso paralelo aos dados, podem ser adicionadas a esse conjunto:

- **open:** solicita a abertura de um arquivo distribuído. Se o arquivo não existir este poderá ser criado;
- **close:** fecha um arquivo distribuído;
- **read:** lê de um arquivo;
- **write:** escreve num arquivo;
- **lseek:** ajusta a posição vigente de um arquivo previamente aberto;
- **rename:** renomeia um arquivo;
- **unlink:** remove um arquivo.

Os parâmetros dessas primitivas podem apresentar variações, porém a maioria dos sistemas busca manter os parâmetros das respectivas primitivas Unix, ocultando a fragmentação e distribuição do arquivo. No entanto parâmetros especiais podem ser adicionados, mesmo sobre essas primitivas elementares, para que o programador faça ajustes específicos do sistema de arquivos, a fim de explorar de maneira mais eficiente a distribuição de dados e o paralelismo das operações.

Dentro do contexto de um sistema sem um processo coordenador de serviços, adotado pelo *framework*, os serviços de abertura, fechamento, renomeação e removeção de um arquivo envolvem a consulta aos Meta-servidores. As exceções são as primitivas de leitura e escrita, que interagem diretamente com os Servidores sem a intermediação

de outros processos. A primitiva *lseek* executa seu serviço diretamente sobre o Cliente, sem a necessidade de utilização da rede de comunicação.

Os serviços relacionados às primitivas *unlink*, *rename* e *close* podem ser realizados de forma síncrona ou assíncrona em alguns sistemas, escolha esta que pode ser feita pelo ajuste dos parâmetros. Embora não ilustrado nesse nível de abstração, essa característica será considerada nas especificações de sistemas a serem usados como estudo de caso. Além das primitivas que solicitam serviços, o sistema conta com outras, para, por exemplo, informar o resultado das solicitações de operações sobre arquivos.

3.1.2 Arquitetura do Framework Conceitual

Esta seção apresenta, em linhas gerais, a linguagem de modelagem unificada UML e a descrição de uma arquitetura básica compartilhada pelos sistemas de arquivos paralelos distribuídos utilizando conceitos dessa linguagem.

3.1.2.1 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) (UML, 2006) é uma linguagem de especificação e modelagem empregada na engenharia de software, padronizada em 1997, quando a *Object Management Group* (OMG) (OMG, 2006), uma associação que promove o uso da tecnologia de orientação a objetos, aceitou a versão 1.1 da linguagem sugerida por um consórcio de empresas de informática. Desde então, novas versões da UML vêm sendo disponibilizadas pela OMG, corrigindo pequenas falhas e adicionando melhorias na versão original. A versão 2.0 da linguagem, cuja primeira parte se tornou disponível desde 2004, é considerada a mais significativa revisão da versão original, incluindo novas elementos de modelagem e diagramas.

A UML possui uma notação gráfica padronizada, que pode ser usada para criar um modelo abstrato de um sistema, denominado modelo UML. Assimila conceitos de orientação a objetos, por ser este o paradigma mais utilizado atualmente no desenvolvimento de *softwares*. Um modelo UML é descrito por um conjunto de

diagramas que representam as partes e os diversos níveis de abstração de um sistema, modelando seus aspectos estruturais e comportamentais. Os Diagramas Estruturais definem a arquitetura de um modelo como sendo composta por classes, objetos, interfaces e componentes físicos que mantém relacionamentos e dependências entre si. Os Diagramas Comportamentais capturam as interações e estados assumidos pelos elementos da estrutura de um modelo durante o decorrer de sua execução.

Neste projeto, a arquitetura de um sistema de arquivos paralelos distribuídos é representada por um diagrama de classes, capazes de descrever, a estrutura de um sistema como um conjunto de classes e de relacionamentos, sem detalhamento dos métodos que realizam as operações do sistema. Uma classe é um bloco de construção de um *software* orientado a objetos e é representada em UML como uma caixa com o nome da classe. No interior da caixa podem ainda ser representados os atributos (propriedades) da classe bem como seus métodos (operações).

Os principais relacionamentos entre classes são classificados como generalização, associação, agregação e composição. Generalizações representam herança, em que uma classe (subclasse) herda as características de outra classe (superclasse). A associação denota o relacionamento entre duas classes e pode indicar o papel que cada classe assume na associação, a cardinalidade entre as classes, a direção e as restrições que são aplicadas no relacionamento. A agregação é uma forma especial de associação, denominada “todo/parte”, usada para indicar que uma classe, denominada de todo, é composta por uma coleção de outras classes que representam suas partes. As instâncias das partes possuem um ciclo de vida independente da instância da sua classe todo. Se existe essa dependência, a associação é denominada composição e indica que, se a instância da classe todo é eliminada da estrutura, todas as instâncias de suas partes também serão.

Cada relacionamento possui uma representação gráfica no diagrama de classes UML. Generalizações são ilustradas como uma ou mais linhas que conectam as subclasses e a superclasse, com um triângulo na extremidade junto à superclasse. Associações são representadas como linhas, opcionalmente indicando o papel de cada elemento no relacionamento e a cardinalidade das instâncias de classes. Nas agregações, a extremidade da linha ligando a classe todo à classe parte, possui, no lado da classe todo, um losango vazio, enquanto que, nas composições, o losango é cheio. Para mais informações sobre a representação gráfica utilizada nos diagramas de classes e demais diagramas UML, recomenda-se consultar os tutoriais (SPARX, 2006; SUN, 2006).

3.1.2.2 Framework Conceitual

A arquitetura básica compartilhada pelos sistemas de arquivos paralelos distribuídos, representada num alto nível de abstração na Figura 10, é descrito em UML através do diagrama de classes apresentado na Figura 11. Duas classes principais são definidas para representar os processos Cliente e Servidor da arquitetura. Esses processos são distribuídos entre os nós de computação de um ambiente distribuído, comunicando-se através de uma rede local, representada no diagrama pela classe Rede de Comunicação.

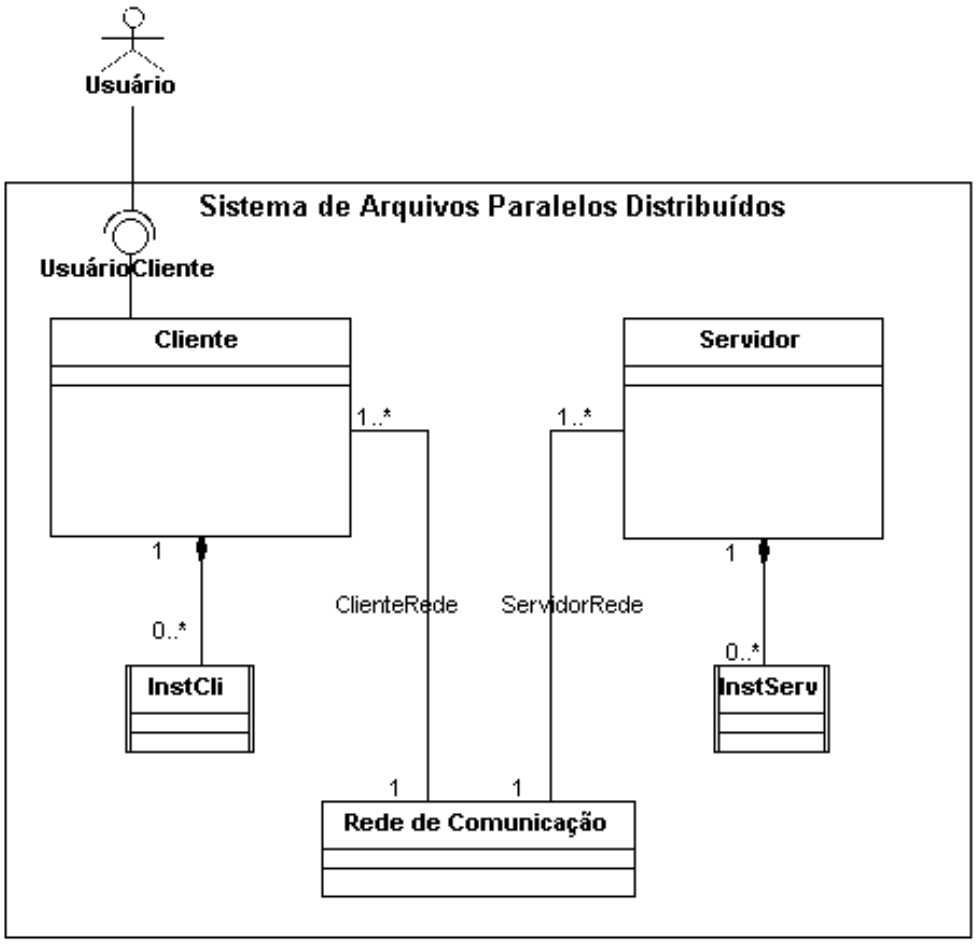


Figura 11: Arquitetura do Framework Conceitual

Os Clientes oferecem primitivas de serviço aos processos Usuários do sistema de arquivos através de uma biblioteca de funções, representada no diagrama de classes pela interface fornecida *UsuárioCliente,* responsável pela definição do conjunto básico de

primitivas descrito na seção 3.1.1. Um Usuário pode se relacionar a apenas um Cliente e é modelado no diagrama como um ator, uma entidade externa que interage com o sistema de arquivos, neste caso, através de uma interface requerida.

Clientes e Servidores são capazes de gerenciar diversos arquivos distribuídos simultaneamente. Para modelar essa característica, classes específicas são definidas para descrever o comportamento, através de MEFEs, das respectivas classes-todo Cliente e Servidor.

A classe *InstCli* especifica a MEFE que realiza controle de arquivos distribuídos no Cliente. Uma instância dessa classe é criada dinamicamente quando uma solicitação de abertura de arquivo é feita pelo Usuário, permanecendo na arquitetura até o fechamento do arquivo. Da mesma maneira, uma instância *InstCli* é criada e permanece ativa enquanto uma operação de renomeação e remoção de arquivo está em andamento.

A associação de composição mostra que as instâncias *InstCli* são partes do Cliente e que seu ciclo de vida está associada ao ciclo da instância da sua classe todo. As instâncias-parte são descritas no diagramas como classes ativas, ou seja, que possuem execução paralela com terminação independente, equivalente aos conceitos de processos ou *threads* de um ambiente de implementação. Num determinado instante, um Cliente pode estar manipulando nenhum ou diversos arquivos, como indica a cardinalidade da composição (0..*).

Analogamente, a classe Servidor é composta por instâncias da classe *InstServ*, as quais cria ou elimina dinamicamente conforme as operações sobre os fragmentos locais dos arquivos distribuídos são solicitadas pelos Clientes. Dessa maneira, uma instância de *InstServ* é criada pelo Servidor para o controle de operações de abertura, renomeação e remoção, sendo eliminada quando o arquivo é fechado ou quando uma operação é concluída.

A comunicação entre as instâncias das classes-parte *InstCli* e *InstServ* são realizadas por meio das instâncias das suas classes todo que utilizam os serviços oferecidos pela *Rede de Comunicação* para esse fim. Um Servidor acessa a Rede de Comunicação através da associação *ServidorRede*, a qual representa diversas instâncias do Servidor conectadas ao meio de comunicação. De maneira equivalente, a associação *ClienteRede* representa a conexão das instâncias do Cliente ao meio. Em outras palavras, as associações possibilitam a um Cliente se comunicar com qualquer instância Servidor e vice-versa.

3.2 Escolha da TDF para a descrição formal do Framework Conceitual

Para possibilitar a adoção de uma técnica de descrição formal no ciclo de desenvolvimento de sistemas de arquivos paralelos distribuídos, o *framework* conceitual, representado pelo diagrama de classes da Figura 11, deve ser descrito segundo os conceitos de projeto de uma das TDFs apresentadas no Capítulo 2. Essa descrição corresponde a uma especificação da arquitetura básica compartilhada pelos sistemas desse domínio de aplicação.

A escolha da TDF deve ser orientada pela semelhança entre seus conceitos arquitetônicos e comportamentais e aqueles encontrados nos projetos de sistemas distribuídos. Em (BARBOSA, 2001) é conduzido um estudo comparativo entre as principais TDFs. A Figura 9, da seção 2.5, demonstra, resumidamente, os conceitos que LOTOS, E-LOTOS, SDL e Estelle dispõem para descrever sistemas.

Estelle e SDL foram concebidas para a especificação de sistemas distribuídos. Por essa razão possuem conceitos próprios para representar as partes, ou componentes, de um sistema (processos e módulos, respectivamente). Comportamentos são descritos através de MEFEs, amplamente empregadas para a descrição dos processos de sistemas de arquivos paralelos distribuídos conjuntamente a tabelas de estados. LOTOS e E-LOTOS, que são independentes de padrões de projeto, não possuem conceitos específicos para a representação de componentes e de comportamentos como MEFEs, o que dificulta a descrição de sistemas distribuídos através dessas TDFs.

Tendo em vista que a especificação será utilizada para gerar uma implementação do sistema, Estelle foi avaliada como sendo a mais adequada para este projeto. As razões da escolha dessa TDF são descritas abaixo (WASSYNG; LAWFORD, 2003; FECKO et al., 2000):

- as principais características e conceitos arquitetônicos de um framework conceitual, necessários para a definição dos sistemas deste projeto, são naturalmente mapeadas para os conceitos e construções Estelle;

- a sintaxe familiar de *Estelle* (próxima a *Pascal*) facilita o seu aprendizado e a compreensão das especificações pelos profissionais atuantes na área de informática, que são os potenciais projetistas e implementadores desse tipo de

sistema. O mesmo não ocorre com a sintaxe de *LOTOS*, que é baseada em modelos algébricos pouco amigáveis, e com a sintaxe de *SDL*, que foi desenvolvida primordialmente para a especificação de sistemas de telecomunicações, sendo bastante difundida junto à comunidade dessa área;

- a maior facilidade para a obtenção semi-automática de uma implementação de um sistema de arquivos distribuídos, a partir de sua especificação formal, foi outro fator decisivo em favor de *Estelle*, uma vez que essa TDF é *implementation-oriented* ao contrário de LOTOS que é *specification-oriented*;

- finalmente a disponibilidade de um conjunto integrado de ferramentas bastante confiável, no caso o *EDT*, também influenciou nessa escolha.

Uma vez definida Estelle como sendo a TDF a ser utilizada na descrição formal do framework conceitual, uma especificação para um sistema de arquivos paralelos básico pode ser construída.

3.3 Especificação Formal do Sistema Básico em Estelle

A descrição formal do sistema básico de arquivos paralelos distribuídos foi realizada utilizando-se conceitos de projeto Estelle mapeados a partir de conceitos UML equivalentes, o que permitiu descrever formalmente a arquitetura do diagrama de classes da Figura 11.

Classes definem componentes de um sistema que apresentam um comportamento bem definido. Em Estelle, o conceito estrutural que cumpre esse requisito é o conceito de *módulo.* As interfaces dos módulos, assim como as de classes, descrevem as interações, ou serviços, realizados pelos módulos. As associações de um diagrama de classes simbolizam a capacidade de uma classe invocar, ou enviar uma mensagem, para a sua classe associada. Essa associação encontra equivalência nos conceitos Estelle de *pontos de interação* e *canais de comunicação*.

Para definir a arquitetura Estelle do sistema, foram criados módulos para representar as classes Cliente, Servidor e Rede de Comunicação do diagrama da Figura

11. Para facilitar a visualização da estrutura de módulos, essa arquitetura inicial representa o sistema apenas com os módulos pertencentes ao mesmo nível hierárquico, sem revelar sua estruturação interna. O refinamento das classes-todo Cliente e Servidor em suas classes-parte, descrito no diagrama de classes como composições, será demonstrado na seção subseqüente.

Um módulo do tipo externo, que define apenas sua interface e representa uma entidade externa que interage com o sistema, foi incluído na arquitetura para representar o Usuário. A Figura 12 representa o sistema de arquivos mapeado para Estelle.

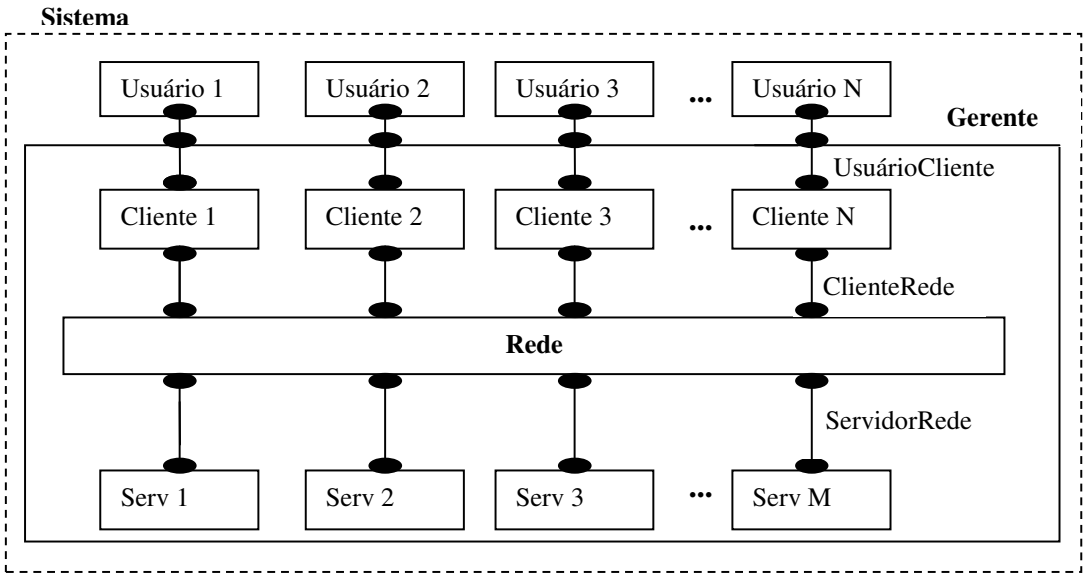


Figura 12: Arquitetura Estelle de um sistema de arquivos paralelos distribuídos

Nessa representação o módulo Sistema (sem transições e sem cabeçalho) engloba todo o sistema. Na sua parte de declaração são definidos os módulos Usuário e Gerente e na sua parte de inicialização são definidas as instâncias estáticas e os *links* entre esses módulos. O módulo Gerente foi incluído para envolver, com exceção dos módulos Usuário, todos os demais módulos, impossibilitando que os Usuários do sistema observem seu interior, tornando assim transparente a realização de serviços pelo protocolo.

Na representação gráfica Estelle não há uma maneira de representar diretamente a cardinalidade entre Clientes, Servidores e a Rede. Por essa razão, são representadas

explicitamente na estrutura Estelle diversas instâncias do módulo Cliente (de 1 a *N* instâncias) e do módulo Servidor (de 1 a *M* instâncias). Apenas uma instância da Rede é declarada, conectando-a aos Clientes e Servidores. Todas essas instâncias são declaradas e inicializadas estaticamente no *Gerente* e seu número permanece inalterado durante o ciclo de vida do sistema.

O módulo Rede possui um comportamento bastante simples. Este possui um único estado e periodicamente inspeciona os seus pontos de interação em busca de mensagens a serem transmitidas. Ao detectar a entrada de uma nova mensagem, a Rede a direciona à instância apropriada do módulo destino (Cliente ou Servidor).

O esqueleto da especificação Estelle, relativa à arquitetura apresentada na Figura 12, é ilustrado na Figura 13. Neste estão presentes os canais de comunicação, os cabeçalhos dos módulos, as inicializações das instâncias estáticas dos módulos Rede, Servidor e Cliente, que são realizadas pelo módulo Gerente, e as inicializações das instâncias estáticas dos módulos Gerente e Usuário, que são realizadas pelo módulo que representa a especificação do Sistema.

Não há determinação explícita do número de Clientes e Servidores do sistema, determinado pelas constantes *N_usuarios* e *N_servidores*, que podem assumir qualquer número inteiro positivo não nulo. A cláusula *timescale* determina a unidade de tempo base do sistema, utilizada na habilitação de transições atrasadas. Embora a especificação associe a contagem do tempo em segundos, essa contagem é, na verdade, assumida como sendo em microssegundos.

Pelo canal *UsuarioCliente* o Usuário solicita os serviços oferecidos através das primitivas descritas na seção 3.1.1 e recebe o resultado das mesmas. Representa o acesso dos Usuários ao sistema via a interface provida/requerida do diagrama de classes. Os demais canais (*ClienteRede* e *ServidorRede*) transportam primitivas internas transparentes aos Usuários e que são responsáveis pela realização das operações. Representam as associações das classes Cliente e Servidor com a Rede de Comunicação. A determinação de primitivas internas depende do comportamento de cada processo e da estratégia seguida pelo projetista para resolver os problemas de localização entre processos dentro de uma arquitetura distribuída.

```
specification Sistema_Basico;

default individual queue;
timescale seconds;

{ parte de declaração do Módulo SAPDD }
const
  N_usuarios =  any integer; {nro de Usuários do Sistema}
  N_servidores =  any integer;  {nro de Servidores do
Sistema }
 Cli = any integer;    { numero maximo de arquivos que
um cliente pode manipular simultaneamente }
  Serv = any integer; { numero maximo de arquivos que
um servidor pode manipular simultaneamente }


...
{ declaração dos canais de comunicação }
channel UsuarioCliente(usuario,provedor);     ... ;
channel ClienteRede(usuario,provedor);   ... ;
channel ServidorRede(usuario,provedor);   ... ;

{ definicao do Usuário ----------------------------- }
module Usuario systemprocess; { cabeçalho do Usuário }
  ip U: UsuarioCliente(usuario);   { p. i. externo }
end;

body CorpoUsuario for Usuario;   { corpo do Usuário }
external;

{ definicao do Modulo Gerente -------------------- }
module Gerente systemprocess; { cabeçalho do Gerente }
 {pontos de interação (p. i.) externos }
 ip UC: array[1..N_usuarios] of UsuarioCliente(provedor)
end;

body CorpoGerente for Gerente; { corpo do Gerente }
  { declaração do Módulo Cliente ---------------- }
  module Cliente process (ind_cli:integer);  { cabeçalho }
    ip C: UsuarioCliente(provedor);  { p. i. externos }
       R: ClienteRede(usuario);
  end;

  body CorpoCliente for Cliente;  { corpo do Cliente }
    ...
  end; { fim da especificação do corpo do Cliente }



{ declaração do Módulo Servidor ----------------    }
module Servidor process (ind_serv:integer);{ cabeçalho }
  ip S: ServidorRede(provedor);   { p. i. externo }
end;

body CorpoServidor for Servidor;  { corpo do Servidor }
```

```
{ declaração do Módulo de Rede ----------------------}
module Rede process;       { cabeçalho da Rede }
{ pontos de interação externos }
ip RC:array[1..N_usuarios] of ClienteRede(provedor);
   RS:array[1..N_Servidores] of ServidorRede(usuario);
end;

body CorpoRede for Rede;   { corpo da Rede }
 ...
end; { Fim da especificação do corpo da Rede }


{ parte de declaração do Módulo Gerente ---------------- }
{ declaração de variáveis de módulo}
  modvar MCliente:array[1..N_usuarios] of Cliente;
         MServidor:array[1..N_servidores] of Servidor;
         MRede:Rede;
  initialize      { criação estática das ocorrência  }
  begin
     init MRede with CorpoRede;    { Módulo de Rede }
    all i:1..N_servidores do
    begin    { Módulos Servidores }
          init MServidor[i] with CorpoServidor(i);
               { conexão dos Servidores com a Rede }
          connect MServidor[i].S to MRede.RS[i]
    end;
    all i:1..N_usuarios do
    begin      { criação dos Módulos Cliente }
         init MCliente[i] with CorpoCliente(i);
                { conexão dos Clientes com a Rede }
          connect MCliente[i].R to MRede.RC[i];
                { vinculação dos Clientes com seus
 respectivos Usuários }
          attach UC[i] to MCliente[i].C;
     end;
   end;
 end; { Fim do Gerente }

{ parte de declaração do Módulo SAPDD---------------- }
{ declaração de variáveis de Módulo }
 modvar MUsuario:array[1..N_usuarios] of Usuario;
          MGerente:Gerente;
 initialize  {criação das ocorrências estáticas do Usuário}
 begin
   { criação do Módulo Gerente }
   init MGerente with CorpoGerente;
   all i:1..N_usuarios do
   begin
        { criação dos Módulos Usuários }
        init MUsuario[i] with CorpoUsuario;
       { conexão dos Usuários com o Gerente }
       connect MUsuario[i].U to MGerente.UC[i];
   end;
 end;

end. { Fim da declaração do Sistema_Basico }
```

Figura 13: Esqueleto da especificação Estelle da arquitetura de um sistema

3.3.1 Refinamento da Arquitetura

A arquitetura da Figura 12, embora revele os principais componentes internos de um sistema de arquivos paralelos distribuído, é uma representação num alto nível de abstração. Como descrito no diagrama de classes da Figura 11, um sistema de arquivos possibilita a manipulação paralela de diversos arquivos.

Na arquitetura UML do *framework* conceitual, a manipulação de vários arquivos pelo Cliente e pelo Servidor é representada, respectivamente, pelas classes *InstCli* e *InstServ*, responsáveis pela execução do comportamento descrito pela MEFE da sua respectiva classe-todo. Em Estelle, uma composição dessas classes-parte é naturalmente mapeada para um conjunto de sub-módulos, cujas instâncias podem ser criadas e removidas dinamicamente e serem executadas em paralelo.

O refinamento da arquitetura inicial, representado na Figura 14, demonstra a estruturação interna dos principais módulos do sistema. A cardinalidade da composição é representada graficamente pelos diversos sub-módulos. Como estes são criados e eliminados dinamicamente, subentende-se que, num determinado instante, um Cliente ou Servidor pode não estar gerenciando nenhum ou diversos arquivos, situação que representa a cardinalidade (0..*) do diagrama de classes. Cada sub-módulo, por pertencer à estrutura interna de seu módulo-pai, somente pode estar associado a um módulo que descreve a classe-todo.

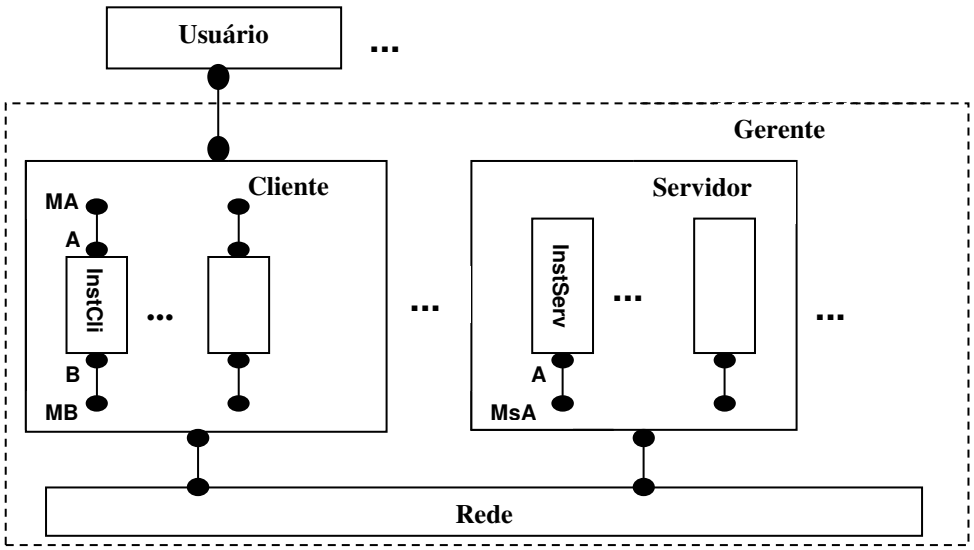


Figura 14: Refinamento dos módulos Cliente e Servidor

Quando necessário o gerenciamento de um determinado arquivo, o módulo pai (Cliente ou Servidor) cria, dinamicamente, uma instância do módulo filho responsável por esse gerenciamento (*InstCli* ou *InstServ*, respectivamente). Quando o mesmo torna-se desnecessário a instância é removida.

Nas próximas seções serão apresentados os esqueletos das especificações dos módulos filhos e sua integração com seu módulo pai. Essas especificações complementam o esqueleto da especificação global do sistema descrito na Figura 13.

3.3.1.1 O Módulo Cliente

A Figura 15 ilustra o esqueleto da especificação do corpo de Cliente, contendo a definição de seu módulo filho, sendo que o comportamento de *InstCli* é descrito através de uma *MEFE*.

Além de controlar a criação e a remoção dinâmicas das *InstCli*, o Cliente gerencia as primitivas enviadas e recebidas por cada uma destas. Como a comunicação entre o Cliente e suas instâncias filhas é realizada através da troca de mensagens, os pontos de interação externos *A* e *B* de cada *InstCli* são conectados aos respectivos pontos de interação internos *MA* e *MB* do Cliente. O número máximo de *InstCli* de um Cliente é igual ao número máximo de arquivos que o seu Usuário pode manipular simultaneamente.

```
body CorpoCliente for Cliente;
    ip  MA: array[1..Cli] of UsuarioCliente(usuario); { pontos de interação internos }
        MB: array[1..Cli] of ClienteRede(provedor);

    { definicão do módulo InstCli -> módulo filho de Cliente }
    module InstCli process (ind_inst, ind_cli:integer);
      ip A: UsuarioCliente(provedor);   { pontos de interação externos }
         B: ClienteRede(usuario);
    end;

    body CorpoInstCli for InstCli;
     {definição da MEFE para o módulo filho do Cliente}
    end; { corpo InstCli }
     ...
   trans
     { criação dinâmica de módulos }
     ...
end;  { CorpoCliente }
```

Figura 15: Esqueleto da especificação do corpo de Cliente

É através das transições de *InstCli* que o sistema local de arquivos é acionado para criar e abrir fragmentos distribuídos como se fossem arquivos comuns. As operações de remoção e renomeação desses fragmentos, bem como a leitura e escrita de dados, são também definidas pelas transições desse sub-módulo.

3.3.1.2 O Módulo Servidor

A Figura 16 ilustra um esqueleto da especificação do corpo de Servidor, contendo a definição de seu módulo filho, sendo que o comportamento de *InstServ* é descrito através de uma MEFE.

Um Servidor cria uma nova instância filha *InstServ*, e estabelece seus *links* de comunicação, sempre que o Cliente lhe solicita uma operação de abertura, remoção ou renomeação sobre um novo arquivo.

Uma instância é eliminada quando o serviço, por esta administrado, é encerrado (na renomeação ou na remoção) ou quando o Cliente solicita que o segmento de arquivo, que esta representa, seja fechado.

```
body CorpoServidor for Servidor;
  ip MsA: array [1..Serv] of ServidorRede(usuario); { pontos de interação interno }

  { definicao dos modulos filhos do Servidor }
  module InstServ process (ind_serv);
    ip A: ServidorRede(provedor); { ponto de interação externo }
  end;

  body CorpoInstServ for Instserv;
    {definição da MEFE para o módulo filho do Servidor}
 end; {CorpoInstServ}
 ...
trans
  { criação dinâmica de módulos }
  ...
end; { CorpoServidor }
```

Figura 16: Esqueleto da especificação do corpo de Servidor

Às transições do módulo Servidor, que controlam o acesso aos seus módulos filhos, é incorporada a MEFE que permite o acesso aos meta-dados armazenados

naquele Servidor. Por se tratar um comportamento simples, representado por apenas um estado onde cada solicitação para alterar ou recuperar meta-dados é prontamente executada e respondida, não há a necessidade de se criar uma instância de módulo apenas para gerenciar essas requisições. Por essa razão não há uma definição específica de uma estrutura para essa funcionalidade na arquitetura do Servidor.

3.4 Complementação da Especificação

O processo de especificação de um sistema de arquivos paralelos distribuídos, baseada na arquitetura do *framework* conceitual apresentado, consiste em tomar como base para a especificação a arquitetura do sistema básico descrito na seção anterior, atribuindo à mesma código Estelle específico, que reflita características próprias do sistema que se pretende descrever formalmente.

A especificação completa de um sistema simples pode ser obtida sem a necessidade de alterações na arquitetura original do *framework* conceitual, ou seja, sem que haja a inclusão de novos componentes na estrutura ou a necessidade de definição de novas operações. Entretanto sendo esse *framework* abstrato por reunir apenas características semelhantes de sistemas que podem adotar diferentes maneiras de executar um mesmo conjunto de operações, algumas decisões de projeto são deixadas a cargo do especificador, que deve preencher a especificação gerada a partir do *framework* com as características particulares do sistema que pretende descrever. As principais decisões a serem tomadas, para a especificação formal de um sistema de arquivos descentralizado utilizando-se a abordagem proposta, são as seguintes:

- *Determinação do comportamento interno dos processos do sistema*: para um framework concebido em Estelle, os comportamentos dos processos devem ser definidos pelo projetista e descritos por meio de MEFEs ;

- *Definição dos canais de comunicação*: o framework fornece na sua arquitetura os canais de comunicação, que permitem a troca de interações entre os módulos. As interações e seus parâmetros correspondentes somente podem ser definidos desde que as MEFEs estejam disponíveis;

- *Definição de funções que armazenam informações internas aos módulos*: os módulos do sistema precisam armazenar informações internas que representam a situação dos arquivos por estes manipulados e que devem ser consultados ou alterados conforme os serviços são realizados. Meta-servidores também precisam de estruturas de dados para gerenciar meta-dados dos arquivos distribuídos;

- *Localização dos módulos na Rede*: a localização dos componentes de um sistema, numa especificação Estelle, é realizada com o auxílio de um número inteiro positivo, que indexa todas as instâncias desses módulos, o que equivaleria ao endereço de rede em um sistema real. O especificador deve prover meios para que os módulos consigam localizar-se dentro do ambiente distribuído;

- *Mecanismo de retransmissão*: os módulos do sistema podem retransmitir solicitações de serviços, caso não obtenham a resposta dentro de um determinado intervalo de tempo. Entretanto a latência da rede ou a sobrecarga no processamento dos módulos podem levar a retransmissões desnecessárias por parte do solicitador da operação. Por essa razão os módulos devem ser capazes de distinguir retransmissões das transmissões. O especificador deve completar o framework com algum mecanismo capaz de fazer essa distinção, impedindo que os módulos refaçam uma operação que já foi realizada, ou receba mais de uma resposta para a mesma solicitação;

- *Estratégia de Distribuição de Dados*: é necessária a descrição do método de fragmentação dos dados, para que possam ser distribuídos entre os Servidores do sistema. Essas operações são especificadas nos módulos Clientes.

3.4.1 Mapeamento de MEFEs para construções Estelle

O comportamento dos componentes de um sistema é representado, dentro do contexto deste trabalho, através de MEFEs. Embora tradicionalmente um modelo não descreva MEFEs, por serem partes específicas de cada protocolo, métodos para mapear o comportamento desses sistemas em código para especificação ou implementação, devem ser empregados para assegurar uma descrição eficiente.

Este projeto assume que o comportamento de um sistema é representado semi-formalmente com o auxílio de Tabelas de Estados. O mapeamento de um MEFE descreve, para cada estado presente na tabela, um conjunto de unidades de comportamento, que reagem ao recebimento de uma interação dentro daquele estado ou reagem à passagem de um determinado intervalo de tempo. Uma unidade de comportamento executa uma ação, que pode enviar uma interação para outro componente ou alterar o estado do comportamento vigente. A Figura 17 demonstra um *template*, do aninhamento de unidades de comportamento, usado para fazer a descrição do comportamento de um componente num *framework*.

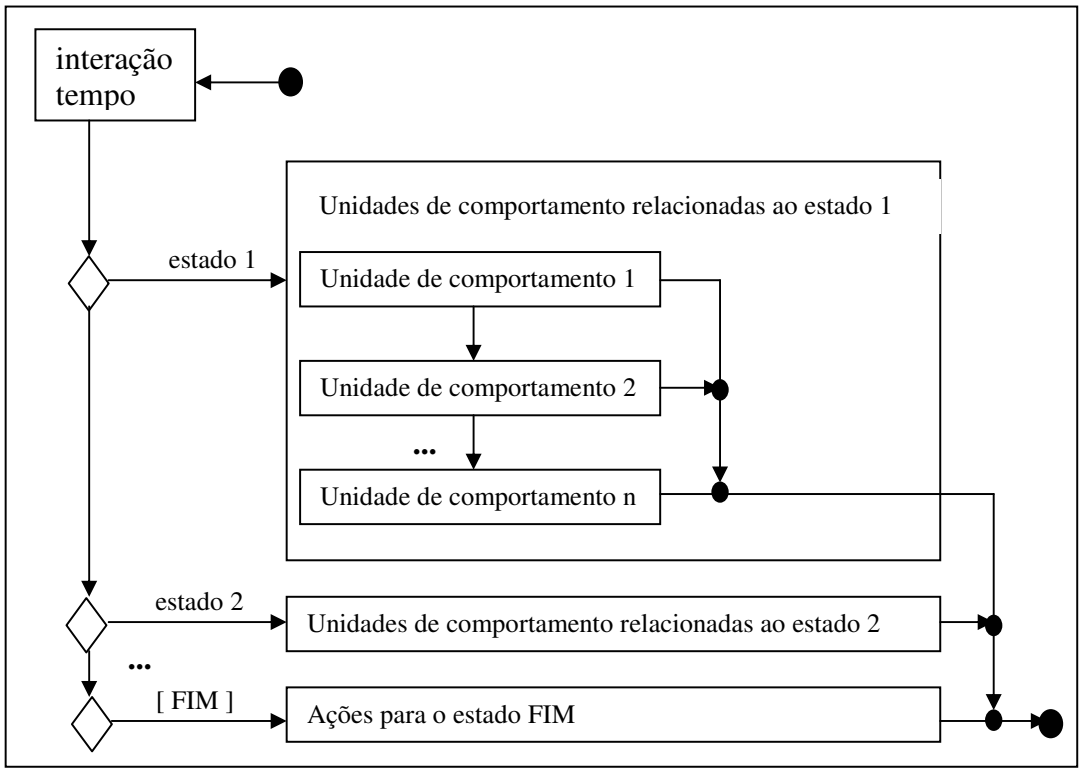


Figura 17: Diagrama de aninhamento de unidades de comportamento

O estado FIM denota um comportamento que deve ser executado, no caso de nenhuma transição estar preparada para absorver uma interação. Sua função é impedir

que o comportamento apresente *deadlock*, ou seja, receber um estímulo externo que não seja capaz de tratar. O estado FIM pode executar um procedimento, por exemplo, para tratar uma exceção ou simplesmente receber uma interação inesperada, para depois descartá-la, sem tomar nenhuma outra providência.

A estrutura das unidades aninhadas pode ser derivada diretamente de uma tabela de estados que representa, para cada unidade, as condições de disparo (incluem o estado, uma transição de entrada e, opcionalmente, uma condição *booleana*), a ação a ser tomada e o próximo estado assumido. A Figura 18 mostra uma Tabela de estados para um comportamento.

Estado Atual	Condição	Ação	Próximo Estado
estado 1	pi.inter	<ações>	estado 2
estado 1	*timeout*	<ações>	estado 2
...			

Figura 18: Tabela de estados para um comportamento

A primeira linha da tabela descreve uma unidade de comportamento disparável a partir do estado *estado1* (primeira coluna), desde que seja recebida uma interação no ponto de interação *pi* (segunda coluna). Já a terceira coluna da primeira linha descreve um conjunto de ações, enquanto que a quarta coluna indica o próximo estado alcançado após a execução do comportamento. A segunda linha descreve um comportamento espontâneo, executado se o estado vigente for *estado1* e transcorrido um determinado tempo (situação indicada pela utilização da palavra *timeout* como condição).

Em Estelle uma Tabela de Estados é descrita na parte de transições de um módulo, através da utilização de um conjunto de conceitos dessa TDF, onde cada unidade de comportamento corresponde a uma transição. A sintaxe Estelle, para a descrição de MEFEs, foi apresentada no Capítulo 2, na seção 2.4. A semântica de execução das transições foi apresentada na seção 2.4.7 desse mesmo capítulo.

Estelle não define uma unidade de comportamento específica para tratar situações não esperadas dentro de um comportamento. Dessa forma, todas as situações devem ser consideradas e descritas no comportamento. Por exemplo, o recebimento de uma interação num estado em que nenhuma transição está preparada para recebê-la, acarretará num *deadlock* daquele comportamento. A transição permanecerá indefinidamente na fila esperando ser consumida, impedindo que novas interações sejam enviadas ao módulo que descreve o comportamento.

3.5 Validação da Especificação

O objetivo da validação de uma especificação é verificar se todos os requisitos e características do sistema, por esta descritos, refletem o que foi definido informalmente ou através de técnicas semi-formais. Uma forma de validação é a simulação, onde o projetista procura detectar erros na especificação por meio de sua execução controlada. Ferramentas como o EDT (EDT, 2000) oferecem ao projetista um ambiente de simulação voltado à análise das propriedades globais da especificação.

Para conseguir executar uma especificação o projetista cria uma entidade, cuja função é interagir com o sistema simulado por meio das interfaces desse sistema. Essa entidade passa então a ser utilizada como *Ponto de Controle e Observação* (PCO), ou seja, um ponto onde é possível controlar o envio de primitivas (entradas) e observar as respostas recebidas (saídas). Dessa forma um teste pode ser realizado enviando-se uma primitiva apropriada ao sistema por meio do PCO e, após o seu processamento pelo sistema, analisando-se o resultado e o comparando com a resposta esperada para aquela situação.

A estratégia adotada para a validação, de um sistema de arquivos distribuídos, compreende a simulação de cada serviço, oferecido sobre os arquivos, com diversas configurações de parâmetros. Isso envolve a análise dos comportamentos das MEFEs dos processos concernidos durante a execução da operação. Dessa forma é possível avaliar tanto o comportamento individual quanto o comportamento conjunto desses componentes.

Além do acesso às interfaces, a simulação pode também acessar a estrutura interna da especificação, o que permite classificá-la como sendo um *teste de caixa branca*. Isso abre ao projetista a possibilidade de acompanhar a execução de um serviço pelo sistema, viabilizando a detecção do ponto onde ocorreu um erro e a definição de sua causa.

3.5.1 Aspectos considerados durante a validação

A estratégia proposta para a validação considera que cada estado das MEFEs de cada processo será alcançado e analisado em pelo menos uma das simulações de serviços. Entretanto há estados responsáveis pela recuperação de erros de transmissão, de falha de processos e de acesso, que são situações que não ocorrem dentro do ambiente ideal oferecido pelos simuladores.

Para contornar esse problema, variáveis especiais podem ser introduzidas na especificação original a fim de auxiliar nas simulações, o que é permitido se o comportamento do sistema não é alterado. Por exemplo, no módulo Rede, variáveis podem fazer o controle do envio e do recebimento de primitivas entre os componentes do sistema. Assim, torna-se possível simular a perda de mensagens pela rede, evitando que uma mensagem seja entregue em seu destino.

Um exemplo de utilização de variáveis especiais é o teste de retransmissão. Quando uma primitiva deixa de ser entregue e não há mais transições para executar, o simulador executa a transição encarregada da retransmissão, que seria disparável se o tempo associado à mesma tivesse expirado. Já para o teste de retransmissão desnecessária, pode-se atribuir um tempo de expiração bastante curto ao *timeout*, que está associado à transição atrasada encarregada da retransmissão.

Além da comunicação é necessário o controle das respostas, geradas por funções primitivas, cuja especificação depende do ambiente de implementação. Por exemplo, no módulo Servidor há funções que acessam o sistema local de arquivos e que não possuem nenhum comportamento na especificação. A inserção de variáveis especiais, nesse módulo, possibilita que essas funções retornem respostas simuladas, em relação ao sucesso ou falha de uma operação sobre um arquivo.

A utilização de novas variáveis e a criação de corpos para módulos *external*, geram uma variante da especificação principal, que é usada unicamente para os propósitos de simulação.

O resultado, que se espera obter da simulação, é a certificação do correto funcionamento das MEFEs, recuperação de falhas do sistema de comunicação e falhas de processos e a ausência de *deadlocks*. Este último é caracterizado na simulação como ausência de transições a serem executadas antes de concluir a realização de um serviço. Além de possíveis problemas no projeto do sistema, erros de especificação também

podem ser detectados. Tanto num caso como no outro, o projetista deve corrigir o problema e refazer a simulação que o detectou.

3.5.2 Cenários de Simulação

Para criar um conjunto o mais abrangente possível de testes, a serem submetidos à especificação durante a simulação, deve-se observar, para cada operação e seus parâmetros, quais transições das MEFEs serão executadas. Restringe-se, dessa maneira, durante um teste, o número de transições a serem verificadas, o que facilita determinar o ponto de ocorrência de erros. Com o mesmo objetivo considera-se, inicialmente, que apenas um Usuário solicita serviços. Posteriormente diversos Usuários podem ser considerados, realizando operações sobre os arquivos de maneira simultânea.

Para cada possível situação, que pode ocorrer nos módulos do sistema devido à execução de um serviço, deve ser criado um traço que representa os estados das MEFEs envolvidas e as interações de entrada e saída. Esses traços são obtidos diretamente dessas MEFEs e descrevem todos os passos da realização de uma operação sobre um arquivo distribuído dentro de um módulo do sistema.

Por exemplo, para a abertura de um arquivo, os traços devem descrever tanto situações em que o arquivo é aberto com sucesso, quanto situações em que problemas podem ocorrer. Considerando-se o módulo Cliente, as situações de operação bem-sucedidas representam aquelas em que não houve falha na rede ou houve falha recuperável por retransmissões. As operações mal-sucedidas são aquelas de falha irrecuperável da rede ou decorrentes de resposta de erro oriundas de um ou mais Servidores envolvidos. Nos Servidores as situações que podem ocorrer são a abertura bem ou mal sucedida de um arquivo.

A combinação de todos os traços gerados em cada módulo gera um conjunto de cenários, que refletem o comportamento do sistema nas mais diversas situações em que este pode se encontrar. Assim sendo, um cenário representa a execução de um serviço sobre um arquivo distribuído, desde a sua solicitação por um Usuário até o retorno da resposta pelo sistema.

3.6 Implementação da Especificação

Após a etapa de validação, um sistema pode ser gerado a partir da especificação com o auxílio de ferramentas, que mapeiam construções Estelle nos blocos de implementação do ambiente de execução. As principais ferramentas criam código C e C++ para essas especificações.

Uma especificação descreve um sistema como um todo, para representar o modo como seus diversos componentes estão interligados e como, em conjunto, realizam uma operação. Porém os sistemas distribuídos possuem processos espalhados por diversos nós de computação, implicando que a especificação, que descreve o sistema como um único bloco, deve ser dividida em especificações separadas, isso para cada processo que irá ser executado num nó diferente. No caso dos sistemas de arquivos distribuídos, a especificação do processo Cliente deve ser criada num arquivo à parte do que contém a especificação do processo Servidor. As funcionalidades do módulo de Rede devem ser distribuídas entre os processos, representando a rede que atenderá ao Cliente e aquela que atenderá ao Servidor.

Ferramentas podem dividir automaticamente a especificação inicial em especificações para cada processo distribuído. Entretanto essa divisão segue a estratégia traçada pela ferramenta. Uma estratégia poderia ser a criação de uma especificação para cada módulo com atribuição de subsistema (*systemprocess* ou *systemactivity*). O problema com essa estratégia é que módulos com atribuição de processos ou atividades (*process* ou *activity*), que fazem parte de um subsistema, também podem ser executados em máquinas distintas, o que levaria a uma divisão errada da especificação. Nessa situação, as opções do especificador seriam alterar a especificação original, tendo em vista a criação automática de especificações separadas para processos distribuídos, ou realizar essa separação manualmente.

Outro aspecto a ser considerado é a implementação das funções primitivas, que envolvem o acesso ao disco pelos Servidores e o acesso ao meio de comunicação. Essas funções devem ser codificadas manualmente na mesma linguagem em que a especificação foi codificada e o código deve ser ligado ao código gerado semi-automaticamente. As funções, que realizam as operações de comunicação, podem ser oferecidas por alguma ferramenta. O projetista pode também optar por implementar suas próprias funções de comunicação, a fim de otimizar o desempenho.

O código para cada processo distribuído é compilado para a geração de código executável. Os processos podem então ser executados nos nós específicos e testes de funcionamento e de desempenho podem ser realizados.

Todo o processo de implementação, as ferramentas utilizadas, as funções que devem ser codificadas manualmente e outros aspectos que devem ser considerados, durante esse processo, são apresentados com detalhes no Capítulo 6.

4 Primeiro Estudo de Caso: Sistema Galley

Neste e no próximo capítulos serão apresentadas as especificações de dois sistemas de arquivos paralelos distribuídos a partir do *framework* conceitual desenvolvido no capítulo anterior, que representam as duas grandes categorias desses sistemas: os que não centralizam as operações e os que possuem um coordenador de serviços centralizado.

Para o estudo de caso do sistema de arquivos descentralizado, será tomado como referência o sistema *Galley* (NIEUWEJARR; KOTZ, 1996), que possui uma arquitetura semelhante à descrita pelo *framework*. O *Network Parallel File System (NPFS)* (GUARDIA; SATO, 1999) foi escolhido como estudo de caso para representar a categoria dos sistemas com centralização de serviços. Para sua especificação será necessário alterar a especificação da arquitetura do *framework* a fim de que um novo componente, responsável pela coordenação das operações sobre arquivos, seja incluído.

Esses dois sistemas foram escolhidos também por compartilharem determinadas características comportamentais, o que significa que terão algumas similaridades nas MEFEs de seus processos, a despeito de pertencerem a categorias distintas. Assumindo algumas simplificações, o NPFS pode ser considerado uma versão centralizada do *Galley*. Desta maneira é possível fazer comparações entre os dois sistemas durante as suas especificações, demonstrando os pontos que devem ser alterados para que se obtenha, a partir da especificação de um sistema descentralizado, um sistema centralizado.

O objetivo é demonstrar a capacidade do *framework* conceitual de sofrer modificações em sua estrutura e comportamento para a descrição de sistemas mais complexos. A inclusão de um novo componente será transparente do ponto de vista dos Usuários do sistema de arquivos, que continuarão interagindo com o novo sistema da mesma forma, ou seja, utilizando o mesmo conjunto de primitivas de serviço. A arquitetura e comportamento do NPFS, como descritos na especificação, podem ser considerados um *framework* para sistemas de arquivos com um coordenador de serviços e, portanto, ser incluído num catálogo de frameworks junto com o framework básico usado para especificar o *Galley*. Nas próximas seções será apresentado o estudo de caso *Galley* com sua arquitetura e comportamento. Será conduzida a especificação formal,

conforme as diretrizes apresentadas no capítulo anterior. O sistema será validado por simulação.

4.1 Especificação e Validação do Sistema Galley

O *Galley* é um sistema de arquivos paralelos distribuídos estruturado como um conjunto de processos Clientes e Servidores, que segue a arquitetura do *framework* conceitual para um sistema básico descrito na seção 3.1.2 do capítulo anterior. Seus processos são:

- Processadores de Computação (*Compute Processors* – CPs): são os processos Clientes associados aos Usuários. Um Usuário no *Galley* é qualquer aplicação que acessa arquivos distribuídos, usando seus serviços, e que é executada num nó de computação. O processo CP, associado ao Usuário, recebe pedidos das aplicações para acesso aos arquivos, converte esses pedidos em solicitações internas do sistema e as passa diretamente aos Servidores apropriados, onde estão sendo executados os IOPs;

- Processadores de E/S (*I/O Processors* – IOPs): correspondem aos processos Servidor do framework.

O comportamento do sistema, determinado pelas MEFEs dos processos e pelas interações trocadas entre os mesmos para a realização de um serviço, será detalhado nas seções subseqüentes.

4.2 Protocolo de Comunicação

Para o sistema ser ativado os processos Servidores devem ser executados em estações de trabalho predeterminadas. Os Clientes podem, por exemplo, receber a

localização dos Servidores na rede através de arquivos de configuração, ficando a cargo do especificador decidir a estratégia a ser adotada nessa ativação.

Concluído esse processo inicial, é possível a qualquer Cliente criar ou abrir um arquivo distribuído. Se a abertura tiver êxito, operações como escrita e leitura podem ser realizadas. A Figura 19 ilustra a interação entre os componentes do sistema na realização de uma operação de abertura de um arquivo distribuído. O estudo das interações entre processos auxilia na determinação de seus comportamentos.

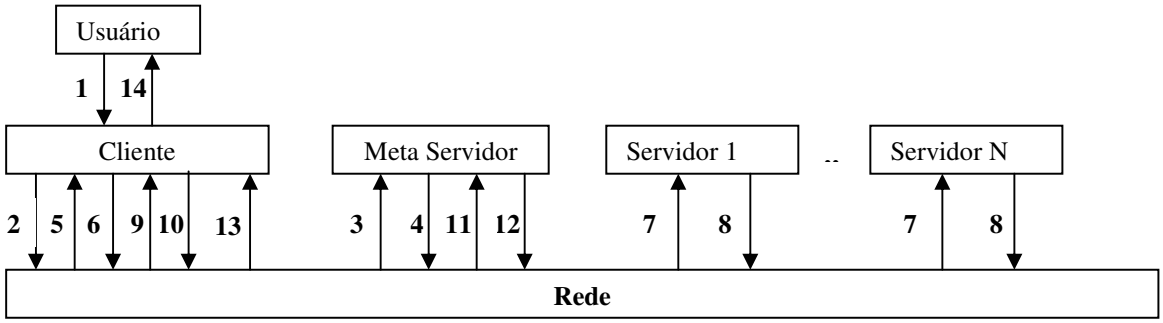


Figura 19: Interação entre processos durante abertura de arquivo

A solicitação parte do Usuário (1) por intermédio de seu processo Cliente (2) correspondente. O Cliente calcula qual Servidor está armazenando as informações (meta-dados), relativas ao arquivo que está tentando abrir. Uma mensagem (3) é enviada ao Meta-servidor que, caso o arquivo exista, responde (4, 5) com os meta-dados daquele arquivo, dentre estes o conjunto de Servidores onde se encontra distribuído.

Com base nessas informações, o Cliente solicita, a todos os Servidores envolvidos (6, 7), que abram seus segmentos locais. Os Servidores, por sua vez, tentam abrir seus fragmentos locais e respondem ao Cliente quanto à solicitação (8). A interação entre Cliente e Servidores pode ocorrer sem problemas (todos os Servidores conseguiram abrir seus fragmentos locais), ou pode ocorrer algum erro (um ou mais Servidores não retornaram ao Cliente com sucesso, ou houve falha na rede). Considerando-se o êxito da operação, o Cliente informa ao Meta-servidor que o arquivo foi aberto (10, 11). O Meta-servidor altera os meta-dados, para registrar a abertura do arquivo, e sinaliza essa alteração ao Cliente (12, 13). Todo o procedimento é transparente para o Usuário, que recebe o resultado final (14) como se estivesse interagindo com um sistema centralizado de arquivos.

Se o arquivo a ser aberto pelo Cliente não existir, este poderá ser criado. Para criar um arquivo distribuído o Cliente informa, na primeira interação com o Meta-servidor, um conjunto de Servidores onde serão armazenados os segmentos desse arquivo. Caso o Cliente não forneça essa informação, o Meta-servidor definirá como padrão um conjunto envolvendo todos os Servidores disponíveis no sistema.

As operações de fechamento e remoção de arquivos levam o Cliente a interagir com os Servidores da mesma maneira que numa operação de abertura. A operação de renomeação necessita de um número maior de interações, tendo em vista que os meta-dados do arquivo renomeado podem migrar de um Meta-servidor para outro, segundo a estratégia de distribuição de informações adotada.

As operações de leitura e escrita envolvem um número menor de troca de mensagens entre os componentes, pois prescindem do acesso aos Meta-servidores. Todos os meta-dados usados por essas operações são recuperados durante a abertura do arquivo que será acessado. Todas as interações entre os processos Cliente e Servidor, para cada operação, são detalhadas nas Tabelas de Estados apresentadas no Anexo A.

A fim de otimizar algumas operações, como a remoção e renomeação de arquivos, o sistema possibilita que algumas operações sejam realizadas de forma síncrona ou assíncrona. No primeiro caso o solicitante do serviço deve aguardar pela conclusão da operação, enquanto que no segundo caso essa espera não é necessária.

Para aumentar o desempenho das operações, os sistemas de arquivos distribuídos tendem a utilizar protocolos de comunicação não orientados a conexão. Como esses protocolos não garantem o recebimento dos dados nem fazem o controle de seqüência, os próprios componentes do sistema de arquivos distribuídos, ou seja, Clientes e Servidores, dispõem de mecanismos que tornam a comunicação confiável. Por exemplo, estes podem usar retransmissões controladas por temporizadores para a recuperação de perdas na rede.

4.3 Primitivas de Serviço

Nesta versão simplificada do *Galley* serão oferecidas apenas as primitivas de serviço, descritas pelo framework, seguidas de seus modos de operação. O prefixo *gfs*

(*Galley File System*) será anexado ao nome de cada primitiva para caracterizá-las como pertencentes a esse sistema .

A primitiva *gfs_open* abre ou cria um arquivo distribuído sobre um conjunto específico de Servidores, sendo que esse arquivo pode ser fechado através da primitiva *gfs_close*. A remoção e renomeação de arquivos são realizadas pelas primitivas gfs_unlink e gfs_rename respectivamente. A leitura e escrita de dados, nos arquivos distribuídos, ficam a cargo das primitivas *gfs_read* e *gfs_write* respectivamente, e a posição vigente de um arquivo pode ser solicitada através da primitiva *gfs_lseek*. Os modos de operação das primitivas são descritos em detalhes via as MEFEs dos processos do sistema, apresentadas na próxima seção, e via Tabelas de Estados, disponibilizadas no Anexo A.

4.4 Comportamento dos Processos

Os processos dos sistemas de arquivos paralelos distribuídos podem ter seus comportamentos definidos através de MEFEs. A elaboração de uma MEFE é realizada ao se analisar os requisitos de cada processo, determinando seus estados e transições conforme seu papel dentro do sistema distribuído. As mensagens enviadas e recebidas por um processo durante a execução de um serviço, tal qual a abertura de arquivo descrita na seção anterior, também auxiliam na elaboração das MEFEs.

Para que a especificação de um sistema de arquivos também possa ser empregada como um modelo, foram definidas MEFEs elementares para os processos Cliente e Servidor, de tal forma que o número de estados fosse o menor possível para cumprir cada comportamento. Essas MEFEs, inspiradas especialmente no estudo do sistema de arquitetura aberta NPFS e *Galley*, podem ser usadas como referencial para a descrição de comportamentos mais elaborados, que reflitam a criação de novas operações ou a inclusão de novos componentes na arquitetura do sistema básico. Dessa maneira, é possível adicionar novos estados e transições, que poderiam atuar sobre os estados já presentes ou sobre os estados recém-adicionados. A Figura 20 apresenta uma MEFE simples, construída por inspeção para o processo Cliente.

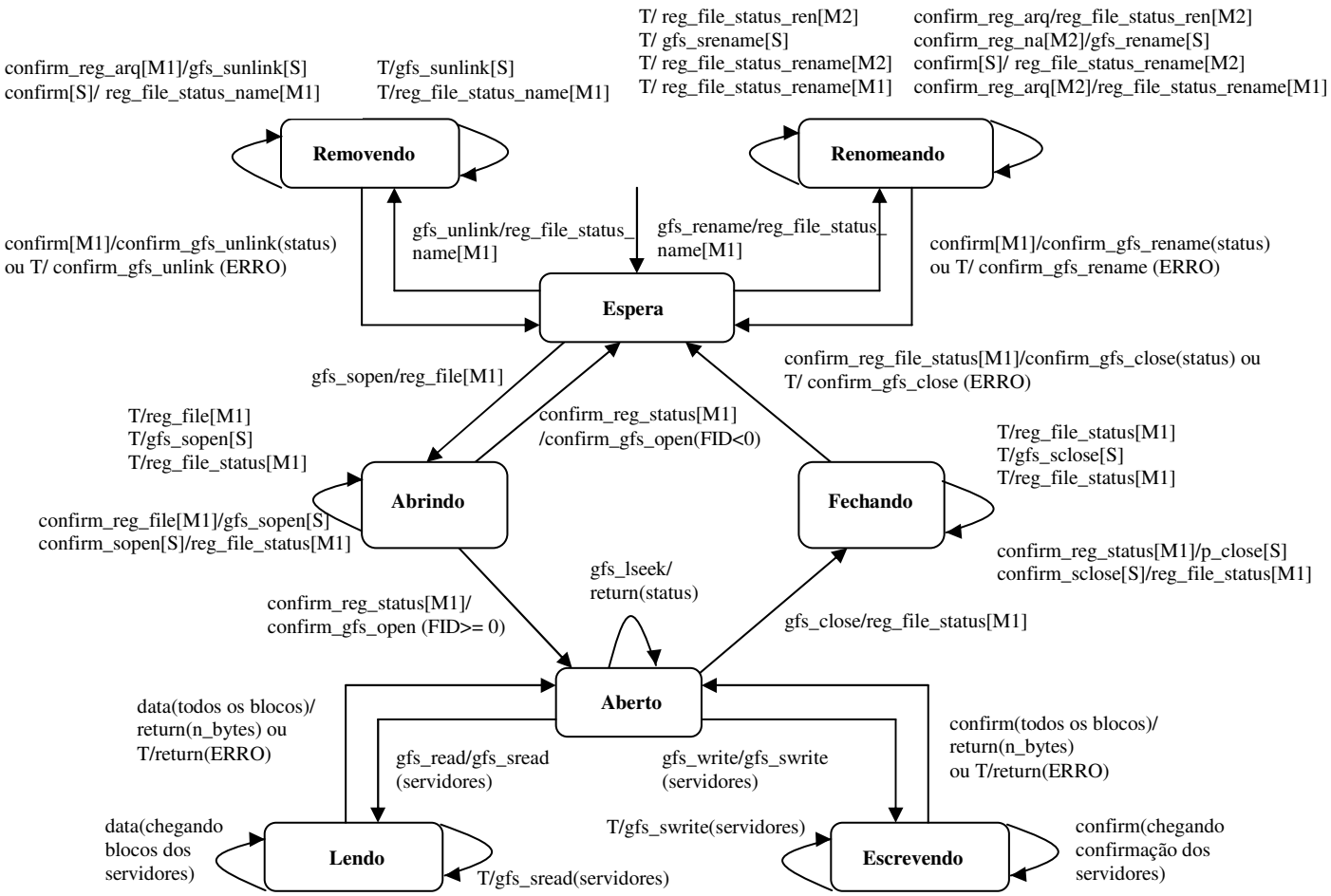


Figura 20: MEFE para o processo Cliente

Quando um estado da MEFE recebe uma interação de entrada, uma interação de saída destinada a outro processo pode ser gerada. Essa situação é ilustrada na Figura 20, com a notação (*interação de entrada/interação de saída*) postada próxima às setas, que indicam o próximo estado que será alcançado devido à interação recebida. Parâmetros podem excepcionalmente ser especificados entre parênteses, para indicar o resultado de uma operação (e.g., *confirm_gfs_open (FID>= 0)*, para indicar abertura de um arquivo com sucesso). O processo, para o qual uma interação é enviada, pode ser especificado entre colchetes (e.g., *confirm_reg_file_status[M1])*, para indicar que a mensagem é destinada a um dos Meta-servidores). Quando não há a especificação do processo, a interação é enviada a um ou mais Servidores.

A MEFE possui basicamente um estado inicial (*Espera*) e um estado para cada operação disponibilizada pelo sistema de arquivos (*Removendo*, *Renomeando*, *Lendo* e *Escrevendo*). O estado *Aberto* foi criado para denotar que o arquivo foi aberto com

sucesso, habilitando o Usuário a solicitar ao Cliente operações de leitura e escrita no arquivo. Para garantir que a MEFE alcance o estado *Aberto* somente no caso da abertura ser bem-sucedida, foi adicionado o estado *Abrindo*. Este estado envia as mensagens de abertura de arquivo a todos os Servidores relacionados com o arquivo e recebe as respostas quanto às solicitações, e também é responsável pela interação com o Meta-servidor, alertando sobre o início e o fim da operação de abertura. O arquivo somente é considerado aberto (denotado na MEFE pela passagem do estado *Abrindo* para *Aberto*), se Servidores e Meta-servidor realizarem suas tarefas com sucesso. Falhas nesses componentes ou falhas de comunicação abortam a operação e a MEFE retorna ao estado inicial (*Abrindo* para *Espera*) sem alcançar o estado *Aberto*.

Para fechar um arquivo o estado *Fechando* foi incluído na MEFE, a fim de realizar a transição entre os estados *Aberto* e *Espera*. O estado *Fechando* solicita o fechamento de arquivo aos Servidores e aguarda as respostas. A MEFE retorna ao estado *Espera* independentemente do sucesso ou falha da operação.

As operações para abrir, fechar, remover e renomear envolvem, além interações com os Servidores, interações com Meta-servidores. Para a elaboração da MEFE do Cliente, considerou-se que um Meta-servidor, relacionado a um determinado arquivo, é encontrado na rede aplicando-se alguma operação que permita localizá-lo pelo nome do arquivo. O uso dessa técnica permite identificar um Servidor associando-lhe um número inteiro e descobrindo a sua localização via uma função que busca distribuir, da maneira o mais balanceada possível, os meta-dados entre os Servidores.

Dessa forma, dependendo do novo nome do arquivo, a operação de renomeação pode exigir que os meta-dados sejam transferidos de um Meta-servidor a outro. Isso significa que o Cliente deve interagir com dois Meta-servidores durante a operação de renomeação de arquivo, o antigo e o novo que irá armazenar os meta-dados uma vez que a operação é concluída, e não apenas com um Meta-servidor como ocorre nas operações de abertura, fechamento e remoção. Todas essas interações estão representadas nas transições da MEFE.

Para a elaboração de uma MEFE, que refletisse o comportamento esperado de um processo Servidor, observou-se que apenas dois estados seriam suficientes. Como estado inicial (*Espera*) também representa a situação na qual o arquivo está fechado, as operações de remoção e renomeação de arquivos podem atuar a partir desse estado. Se a operação for a abertura de arquivo, e sendo esta realizada com sucesso, o estado *Aberto* é alcançado. Nesse estado são realizadas operações de leitura e escrita de arquivo, além

da operação de fechamento que reconduz a MEFE ao estado inicial do Servidor. A Figura 21 demonstra a MEFE descrita anteriormente.

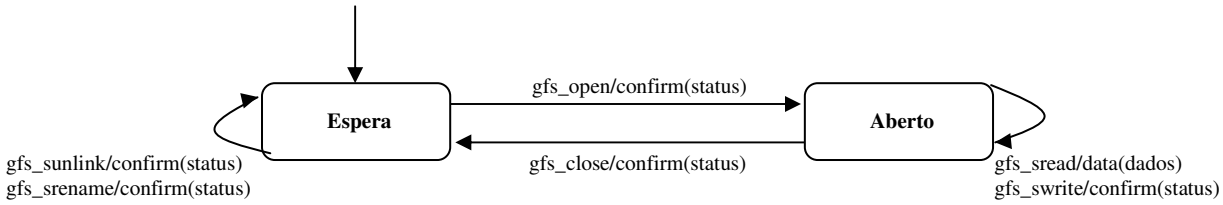


Figura 21: MEFE para o processo Servidor

O mesmo Servidor, que é usado para armazenar e recuperar dados de arquivos distribuídos, também atua como Meta-servidor, significando que o processo precisará responder às solicitações de consulta e modificação em sua base de meta-dados. Essas operações são simples de serem concretizadas, bastando ao Meta-servidor responder prontamente a cada solicitação, sem a necessidade de alterar seu estado inicial para outro estado. A Figura 22 mostra uma MEFE, elaborada para um Meta-servidor, que permite alterar, informar o *status* ou o nome de um arquivo distribuído, ou ainda retornar meta-dados ao Cliente.

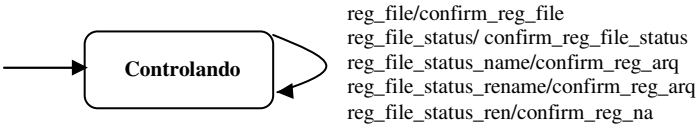


Figura 22: MEFE para um Meta-servidor

4.5 Mapeamento das MEFEs para construções Estelle

Para ilustrar o mapeamento das MEFEs, apresentadas na seção anterior, para construções Estelle, será usado como exemplo a MEFE do módulo Servidor, já que esta possui o menor número de estados. O procedimento emprega o *template* elaborado no Capítulo 3, seção 3.4.1, que parte da Tabela de Estados derivada diretamente da MEFE do módulo. A Figura 23 exemplifica, de forma simplificada, o comportamento dos sub-

módulos do módulo Servidor, responsáveis pela manipulação dos segmentos locais de um arquivo distribuído.

Estado Atual	Evento	Ação	Próximo Estado
Espera	gfs_sopen()	• Abre/Cria segmentos locais • Responde ao Mestre: CONFIRM(STATUS)	Aberto
	gfs_sunlink()	• Remove segmentos locais • Responde ao Mestre: CONFIRM(STATUS)	Espera
	gfs_srename()	• Renomeia segmentos locais • Responde ao Mestre: CONFIRM(STATUS)	Espera
Aberto	gfs_sread()	• Lê bloco de dados solicitado • Envia dados para o cliente: DATA	Aberto
	gfs_swrite	• Escreve bloco de dados recebido: CONFIRM(STATUS)	Aberto
	gfs_close()	• Fecha segmentos • Responde ao Mestre: CONFIRM(STATUS)	Espera

Figura 23: Tabela de Estados do processo Servidor

A Tabela de Estados é naturalmente traduzida para uma estrutura de interações Estelle, conforme demonstra a Figura 24. Nesta figura as interações são recebidas e enviadas pelas transições via ponto de interação A.

```
trans
    from ESPERA to ABERTO
    when A.p_open(...)
    { Abrir segmento local }
    begin
       Abre_CriaSegmentosLocais(...)
       output A.confirm(...)
    end;

    from ESPERA to same
    when A.p_unlink(...)
    { Remover segmento local }
    begin
       RemoveSegmentosLocais(...)
       output A.confirm(...)
    end;

    from ESPERA to same
    when A.p_rename
    { Renomear segmento local }
    begin
       RenomeiaSegmentosLocais(...)
       output A.confirm(...)
end;
```

```
from ABERTO to same
   when A.p_read(...)
   { Ler do segmento local }
   begin
      LeBlocoSolicitado(...);
      output A.data(...)
   end;

   from ABERTO to same
   when A.p_write(...)
   { Escrever no segmento local }
   begin
      EscreveBlocoRecebido(...)
      output A.confirm(...)
   end;

  from ABERTO to ESPERA
  when A.p_close(...)
  begin
     { Fechar segmento local }
     FechaSegmentosLocais(...)
     output A.confirm(...)
        end;
```

Figura 24: Interações da Tabela de Estados do Servidor

As funções, responsáveis pela realização de cada serviço, são declaradas primitivas, uma vez que as suas implementações dependem do ambiente de execução do sistema.

4.6 Definição das interações entre processos

Para que um módulo possa interagir com seu ambiente, deve-se definir um canal de comunicação que seja capaz de conduzir todas as primitivas de serviço especificadas por suas MEFEs. Para o exemplo do módulo Servidor, um ponto de interação, denominado A, deve estar associado a um canal de comunicação. Esse canal, denominado *ServidorRede*, é especificado simplificadamente como:

```
channel ServidorRede(usuario,provedor);
by usuario: gfs_open(...);
            gfs_close(...);
            gfs_unlink(...);
            gfs_rename(...);
            gfs_read(...);
            gfs_write(...);
by provedor: confirm (status);
             data(...);
```

De acordo com essa definição, o ponto de interação *A* assume o papel de provedor no Servidor. O mesmo procedimento deve ser adotado para a declaração dos demais canais de comunicação, que estão associados aos pontos de interação de outros módulos.

4.7 Definição de funções que armazenam informações internas aos módulos

A fim de manter o controle sobre as funções, os módulos devem definir estruturas de dados capazes de armazenar informações internas, bem como um conjunto de procedimentos para manipular e acessar esses dados. No caso do Servidor, uma estrutura foi criada para manter informações, tais como o fid e o *status* dos arquivos distribuídos gerenciados localmente. Sempre que uma solicitação de abertura, renomeação ou remoção de arquivo chega ao Servidor, uma instância para o controle do

arquivo é criada e as informações relacionadas ao arquivo são armazenadas. Uma das razões para esse procedimento é que essas informações devem ser consultadas, a fim de que o Servidor localize entre as instâncias aquela que deve realizar uma operação de leitura, escrita ou fechamento de arquivo, intermediando dessa forma as instâncias com os demais componentes do sistema.

O mesmo controle deve ser mantido sobre os meta-dados. Para gerenciá-los, uma estrutura, que mantém informações sobre os arquivos distribuídos (e.g, nome, fid, tamanho da unidade de distribuição, Servidores envolvidos), é mantida pelo Servidor. Esses meta-dados podem ser consultados pelos Clientes através de primitivas enviadas aos Meta-servidores, as quais implementam funções que retornam as informações solicitadas. Há também funções que alteram os meta-dados na medida em que os arquivos distribuídos são criados, removidos ou renomeados.

Todas as funções, para o gerenciamento de dados nos módulos Cliente e Servidor, estão descritas na especificação completa do sistema, que está disponível no Anexo C.

4.8 Localização dos módulos na Rede

Todos os módulos Cliente e Servidor, que apresentam mais de uma instância no sistema, possuem um número inteiro positivo que permite identificá-los dentro da estrutura. Esses números devem acompanhar as mensagens enviadas pelos Clientes, para que o módulo de Rede possa encaminhá-las ao módulo Servidor destinatário. Essa também é a forma pela qual os Servidores localizam os Clientes, que solicitaram os serviços, para poder-lhes retornar a resposta.

Num sistema de arquivos descentralizado, um dos maiores problemas é distribuir os meta-dados da maneira o mais balanceada possível e localizar os Meta-servidores. Neste tipo de sistema será usada a técnica de *hashing*, onde uma função retorna um número inteiro, baseando-se no nome de um arquivo distribuído, que identifica o Meta-servidor responsável pelos meta-dados do arquivo.

O uso de uma função *hash*, para a localização de Meta-servidores, é bastante comum (GARCIA et al., 2002; GIBSON et al., 1995; NIEUWEJARR; KOTZ, 1996).

4.9 Mecanismo de retransmissão

Com o objetivo de evitar problemas com retransmissões, um mecanismo de controle foi incorporado nos módulos Cliente e Servidor do sistema. Esse mecanismo enumera todas as mensagens trocadas através da rede.

Quando o módulo Servidor recebe uma solicitação de abertura, fechamento, renomeação ou remoção de um arquivo distribuído, o número da mensagem que carrega a solicitação é verificado, para averiguar se não se trata de retransmissão para um serviço já realizado. Em caso afirmativo, a resposta enviada ao solicitador é recuperada de um *buffer*, que armazena as respostas mais recentes geradas pelo seu módulo correspondente. Essa estratégia garante que a resposta enviada, para um mesmo serviço, seja sempre a mesma.

Devido à latência de rede ou ao atraso causado pela sobrecarga de processamento dos módulos, um serviço pode ser indevidamente solicitado novamente, levando a retransmissões desnecessárias das respostas. Os módulos possuem em suas MEFEs transições especiais, que descartam interações recebidas fora do contexto assumido pelo módulo num determinado instante. Entretanto nem sempre a MEFE está disponível para realizar esse controle. Por exemplo, quando um arquivo distribuído é fechado com sucesso, o módulo filho que especifica a MEFE responsável pelo seu gerenciamento é eliminado. Mensagens atrasadas podem chegar destinadas a esse sub-módulo, que não se encontra mais presente na estrutura. Neste caso o módulo pai assume o papel de identificar e eliminar mensagens atrasadas indevidas, destinadas a módulos filhos inexistentes.

Como existe mais de uma maneira de especificar o controle de retransmissão, por exemplo em relação ao modo de enumeração de mensagens e ao tamanho e controle do *buffer* de armazenamento, esse controle não é parte integrante do framework.

4.10 Estratégia de Distribuição de Dados

Na especificação de um sistema de arquivos pode-se deixar a cargo do implementador a escolha de como os dados de um arquivo serão distribuídos entre os

Servidores, deixando o sistema o mais genérico possível em relação a essa questão. Entretanto, para a criação de um sistema de arquivos paralelos completo visando a simulação, a descrição da estratégia de distribuição de dados entre os Servidores pode ser incorporada à especificação do sistema.

O modo de distribuição *round-robin* foi o adotado no sistema *Galley*, por ser o mais comumente empregado nos sistemas de arquivos distribuídos (NIEUWEJARR; KOTZ, 1996). Uma visão geral de seu funcionamento foi apresentada no capítulo 2, na seção 2.1. Os detalhes serão demonstrados durante a validação da especificação do sistema (seção 6.1.10.3), onde será descrita a forma pela qual os dados são divididos nos blocos a serem distribuídos pelos Servidores.

4.11 Validação do Sistema Descentralizado

A validação do sistema segue a estratégia descrita em linhas gerais no Capítulo 3, seção 3.5. Será criada uma entidade que interage com o ambiente simulado através de suas interfaces. Essa entidade é usada como *Ponto de Controle e Observação(PCO)*, ou seja, um ponto onde é possível controlar o envio de primitivas (entradas) e observar as respostas recebidas (saídas). Para cada situação prevista, um teste pode ser realizado via PCO: uma primitiva adequada é enviada ao sistema, esta é processada pelo mesmo e a resposta do sistema é comparada à esperada para aquela situação.

Nos testes, o módulo Usuário é uma interface para o sistema, sendo possível selecionar os serviços a serem realizados através do envio de primitivas ao Cliente associado, e inspecionar as respostas do sistema ao Usuário. Inicialmente foram testados todos os serviços envolvendo um único Usuário e um número variável de módulos Servidor para, em seguida, realizar os testes envolvendo diversos módulos Usuário, que podem solicitar diferentes serviços simultaneamente.

Variáveis especiais foram usadas para o teste de retransmissão devido à falha na rede, ou retransmissões desnecessárias devido a *timeout* muito curto associado à transição responsável pela retransmissão. Também foram inseridas variáveis para controlar a resposta devolvida pelos Servidores, quando há solicitação de serviços.

4.11.1 Ambiente de Simulação

Para a validação das especificações *Estelle* dos dois sistemas, foi utilizado o conjunto integrado de ferramentas *EDT*, constituído pelo compilador *Ec* e pelo simulador/depurador *Edb*, sendo que este último tem três formas principais de atuação:

- análise e validação das propriedades globais da especificação, através de sua execução em um ambiente simulado (CATRINA, I; CATRINA, O, 1998); (DEPLANCHE; OLLIVE; TRINQUET, 1998);

- indicação de erros e de situações de alerta, quando a condição de uma transição é avaliada ou quando a mesma é executada;

- definição de restrições temporais para o sistema, tendo em vista o seu ambiente de execução, o que torna possível estudar o desempenho do sistema simulado (BORCOCI, 1998).

O *Edb* possui um conjunto de comandos, definindo uma linguagem própria, que permitem o acompanhamento passo a passo da execução das transições, no caso de simulação interativa, assim como a definição de *observadores* e *macros*, no caso de simulação automática.

Observadores monitoram determinadas características do sistema e podem interromper a simulação, se ocorrer uma condição pré-estabelecida pelo Usuário. Macros são empregadas na definição de *cenários de simulação*, que representam situações assumidas pelo sistema durante a execução dos seus serviços. As macros podem armazenar informações em arquivos, tais como identificação das transições executadas, interações enviadas ou recebidas e valores de variáveis dos módulos, que permitem, na ocorrência de erros, a recuperação dos passos de execução dos serviços.

4.11.2 Cenários de Simulação

Para a validação de uma especificação *Estelle* devem ser definidos cenários de simulação para os serviços oferecidos pelo sistema em questão. Para cada operação, o comportamento apresentado pelos processos envolvidos deve ser analisado, de forma que possa ser comparado ao comportamento previsto durante a fase de projeto do sistema.

Assim sendo, para cada possível situação que possa ocorrer nos módulos Cliente e Servidor devido à execução de um serviço, foi criado um traço. A combinação de todos os traços, gerados em cada módulo, produziu um conjunto de cenários que refletem o comportamento do sistema nas mais diversas situações em que este pode se encontrar. O objetivo é testar pelo menos uma vez todas as transições envolvidas com cada uma das operações oferecidas pelo sistema.

Cada operação pode ser bem ou mal sucedida na realização de seu serviço. No primeiro caso todos os módulos envolvidos conseguem realizar suas funções porque não ocorrem problemas ou, na presença de algum problema (e.g., falha da rede durante o serviço) este pode ser contornado. No segundo caso, a falha pode ser ocasionada por mais de um motivo alheio ao sistema, como por exemplo a falha de um módulo em contatar outro módulo (ocorrência de erro na rede que não pôde ser resolvido com retransmissões) ou a recusa de um Servidor em realizar o serviço devido a algum problema interno.

A fim de exemplificar o método de validação, as Figuras 25 e 26 ilustram, respectivamente, os *traços* obtidos durante as tentativas bem e mal sucedida de abertura de um arquivo.

Nessas figuras estão representadas as *situações* que podem ocorrer nos módulos Cliente e Servidor, além do Meta-servidor (*Sitn_Cli*, *Sitn_serv* e *Sitn_Meta*). O primeiro conjunto de parênteses contém o estado vigente do módulo e pode ser acompanhado de uma interação de entrada, enquanto que o segundo contém o próximo estado e pode ser acompanhado de uma interação de saída. Se não houver interação de entrada a transição é espontânea. Os conteúdos dos colchetes *[U]*, *[C]* , *[S]* ou *[Meta]* indicam de (ou para) que módulo foi enviada a interação. Um cenário é uma composição dessas situações e pode ser testado através de uma macro específica.

Na Figura 25 as situações do Cliente foram divididas em três subconjuntos. O primeiro subconjunto descreve as situações em que o Cliente envia uma mensagem ao Meta-servidor para registrar o início de uma operação. O segundo representa as situações decorrentes da solicitação da abertura de arquivo aos Servidores envolvidos. O terceiro subconjunto descreve as possíveis situações do Cliente ao solicitar ao Meta-servidor o registro do término da operação.

Cada um desses subconjuntos é constituído de duas situações, uma para descrever o acesso direto aos processos que interagem com o Cliente durante a execução da operação e a outra para descrever o acesso após um determinado número de retransmissões. No Meta-servidor as duas situações representam os registros com sucesso, do início e do término da operação de abertura de arquivo, solicitados pelo Cliente. O Servidor possui apenas uma situação, na qual a abertura local do segmento do arquivo é realizada sem problemas.

A combinação das situações dos processos levou ao conjunto de cenários possíveis para a realização com sucesso da operação. Dentre estes o *Cenário 1* é o mais simples, sendo que a sua simulação tem início com a execução de uma macro que ativa um Usuário, para que o mesmo inicie o serviço de abertura de um arquivo, enviando uma primitiva *reg_file* ao Meta-servidor. Em seguida a simulação é conduzida pela macro, que ativa os módulos Servidor e a simulação é encerrada somente quando a resposta ao serviço solicitado é retornada ao Usuário. Essa resposta é examinada para verificar se é a esperada, sendo que variáveis de módulos também podem ser vistoriadas após a realização do serviço. A escolha das transições para a execução é realizada, de forma indeterminística, pelo próprio simulador.

Testes de operação de Abertura de Arquivo bem sucedidas

Situações no Cliente:

Cliente tenta contactar Meta Servidor para registrar início da operação

Sit1_Cli: Cliente contacta Meta Servidor para registrar início de operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_open[S]
Sit2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta Servidor até conseguir resposta
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) _(ABRINDO) reg_file[Meta]→ ... → (ABRINDO) _(ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

Sit3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta]
Sit4_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta]

Cliente tenta contactar Meta Servidor para registrar término da operação

Sit5_Cli: Cliente contacta Meta Servidor para registrar término de operação
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta] → (ABRINDO) confirm_reg_status_OK[Meta](ABERTO) confirm_gfs_open_OK [U]
Sit6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta] → (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →(ABRINDO) confirm_reg_status_OK[Meta](ABERTO) confirm_gfs_open_OK [U]

Situações no Meta Servidor

Sit1_Meta: Meta Servidor registra início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_OK[C]
Sit2_Meta: Meta Servidor registra término de operação de abertura de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

Sit1_Serv: Servidor abre arquivo com sucesso
(ESPERA)gfs_sopen[C](ABERTO) confirm_sopen_OK[C]

Cenários que levam à abertura de arquivo com sucesso

Cenário 1: (Sit1_Cli ; Sit3_Cli ; Sit5_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 2: (Sit1_Cli ; Sit4_Cli ; Sit5_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 3: (Sit1_Cli ; Sit3_Cli ; Sit6_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 4: (Sit1_Cli ; Sit4_Cli ; Sit6_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 5: (Sit2_Cli ; Sit3_Cli ; Sit5_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 6: (Sit2_Cli ; Sit4_Cli ; Sit5_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 7: (Sit2_Cli ; Sit3_Cli ; Sit6_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv
Cenário 8: (Sit2_Cli ; Sit4_Cli ; Sit6_Cli) → (S1_Meta ; S2_Meta) → Sit1_Serv

Figura 25: Cenários da tentativa bem sucedida de abertura de arquivo

Os cenários mais freqüentes, que levam à falha na tentativa de abertura de um arquivo, são os descritos na Figura 26. Os fatores que determinam essa falha são a incapacidade de um módulo de se comunicar com outro, mesmo após *n* retransmissões para contato (onde $n > 0$ é dependente da implementação) e a incapacidade do Servidor de abrir o arquivo. Nesses cenários a preocupação maior foi testar os módulos responsáveis pelos controles locais dos arquivos, cujos comportamentos refletem as MEFEs apresentadas nas Figuras 20, 21 e 22.

O processo Cliente possui agora situações onde o acesso ao Meta-servidor, ou aos Servidores, pode ser conseguido após uma ou mais tentativas de contato, porém a resposta quanto ao acesso aos meta-dados apresentará algum problema, impossibilitando a consumação do serviço. Também está representada a situação onde o número de retransmissões é esgotado, o que poderia significar falha irrecuperável da rede.

Testes de operação de Abertura de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta Servidor para registrar início da operação

Sit1_Cli: Cliente contacta Meta Servidor para registrar início de operação, que retorna ERRO
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_ERRO[Meta](ESPERA) confirm_gfs_open_ERRO [U]
Sit2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta Servidor até conseguir resposta. Meta Servidor retorna ERRO para a operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) _(ABRINDO) reg_file[Meta]→ ... → (ABRINDO) _(ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_ERRO[Meta](ESPERA) confirm_gfs_open_ERRO [U]

Sit3_Cli: Cliente retransmite pedido, mas não consegue contactar o Meta Servidor
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) _(ABRINDO) reg_file[Meta]→ ... → (ABRINDO) _(ABRINDO) reg_file[Meta] → (ABRINDO)_ (ESPERA) confirm_gfs_open_ERRO [U]
Sit4_Cli: Cliente contacta com sucesso Meta Servidor para registrar início de operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sclose[S]
Sit5_Cli: Cliente retransmite pedido de registro de início de operação para o Meta Servidor até conseguir resposta
(ESPERA) gfs_open[U](ABRINDO) reg_file[Meta] → (ABRINDO) _(ABRINDO) reg_file[Meta]→ ... → (ABRINDO) _(ABRINDO) reg_file[Meta] → (ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sclose[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

Sit6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_ERRO[S](ABRINDO) confirm_gfs_open_ERRO [U], reg_file_status[Meta] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
Sit7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_ERRO[S](ABRINDO) confirm_gfs_open_ERRO [U] , reg_file_status[Meta] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
Sit8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta. Meta Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) _(ABRINDO) confirm_gfs_open_ERRO [U] , reg_file_status[Meta] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
Sit9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta]
Sit10_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(ABRINDO) confirm_reg_status_OK[Meta](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta]

Cliente tenta contactar Meta Servidor para registrar término da operação

Sit11_Cli: Cliente contacta Meta Servidor para registrar término de operação
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta] → (ABRINDO) confirm_reg_status_ERRO[S](ABERTO) confirm_gfs_open_ERRO [U]

Sit12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta] → (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →(ABRINDO) confirm_reg_status_ERRO[S](ABERTO) confirm_gfs_open_ERRO [U]
Sit13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [Meta] → (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [Meta] →(ABRINDO)_(ESPERA) confirm_gfs_open_ERRO [U]

Situações no Meta Servidor

Sit1_Meta: Meta Servidor registra com sucesso início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_OK[C]

Sit2_Meta: Meta Servidor registra com erro início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_ERRO[C]
Sit3_Meta: Meta Servidor registra término de operação de abertura de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

Sit1_Serv: Servidor abre arquivo com sucesso
(ESPERA)gfs_sopen[C](ABERTO) confirm_sopen_OK[C]

Sit2_Serv: Servidor falha ao abrir arquivo
(ESPERA)gfs_sopen[C](ESPERA) confirm_sopen_ERRO[C]

Cenários que levam à abertura de arquivo com sucesso

Cenário 1: (Sit1_Cli)
Cenário 2: (Sit2_Cli)
Cenário 3: (Sit3_Cli)
Cenário 4: (Sit4_Cli ; Sit6_Cli) → (Sit3_Meta) → (Sit2_Serv)
Cenário 5: (Sit4_Cli ; Sit7_Cli) → (Sit3_Meta) → (Sit2_Serv)
Cenário 6: (Sit4_Cli ; Sit8_Cli) → (Sit3_Meta)
Cenário 7: (Sit4_Cli ; Sit9_Cli ; Sit11_Cli) → (S1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 8: (Sit4_Cli ; Sit9_Cli ; Sit12_Cli) → (S1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 9: (Sit4_Cli ; Sit9_Cli ; Sit13_Cli) → (S1_Meta) → (Sit1_Serv)
Cenário 10: (Sit4_Cli ; Sit10_Cli ; Sit11_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 11: (Sit4_Cli ; Sit10_Cli ; Sit12_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 12: (Sit4_Cli ; Sit10_Cli ; Sit13_Cli) → (Sit1_Meta) → (Sit1_Serv)
Cenário 13: (Sit5_Cli ; Sit6_Cli) → (S2_Meta) → (Sit2_Serv)
Cenário 14: (Sit5_Cli ; Sit7_Cli) → (S2_Meta) → (Sit2_Serv)
Cenário 15: (Sit5_Cli ; Sit8_Cli) → (S2_Meta)
Cenário 16: (Sit5_Cli ; Sit9_Cli; Sit11_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 17: (Sit5_Cli ; Sit9_Cli; Sit12_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 18: (Sit5_Cli ; Sit9_Cli;;Sit13_Cli) → (Sit1_Meta) → (Sit1_Serv)
Cenário 19: (Sit5_Cli ; Sit10_Cli ; Sit11_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 20: (Sit5_Cli ; Sit10_Cli ; Sit12_Cli) → (Sit1_Meta ; Sit3_Meta) → (Sit1_Serv)
Cenário 21: (Sit5_Cli ; Sit10_Cli ; Sit13_Cli) → (Sit1_Meta) → (Sit1_Serv)

Figura 26: Cenários da tentativa mal sucedida de abertura de arquivo

Para simular a falha da operação somente após o registro no Meta-servidor, ou após o contato com os Servidores, o Cliente também apresenta situações onde essas operações intermediárias são realizadas com sucesso, o que permite conduzir a simulação até o ponto desejado para se avaliar um determinado problema. Com essa finalidade, o Meta-servidor possui situações de êxito, para a manipulação de meta-dados no início da operação, e o Servidor pode retornar erro em relação à solicitação do Cliente, como também pode responder afirmativamente ao serviço.

Para a situação de erro foram obtidos 21 cenários de simulação que testam desde falhas ocorridas no início da operação, até falhas que se apresentaram quando a operação estava sendo encerrada.

Existem outros fatores que levam à falha, tais como, a tentativa de um Usuário de ler/escrever num arquivo que não foi aberto, ou a tentativa de remover um arquivo que foi aberto por outro Usuário. Para esse tipo de cenário foram criadas macros, que testam a solicitação inadequada de serviços por parte do Usuário. Embora não ilustrados, vários outros tipos de cenários foram concebidos, as macros correspondentes foram criadas, e estes foram testados "exaustivamente".

A Tabela 1 mostra o número de cenários de simulação obtido para cada primitiva de serviço oferecida pelo sistema. A descrição de todos os cenários está apresentada no Anexo B.

	Cenários de Simulação		
Operações	Bem Sucedidas	Mal Sucedidas	Total
Abrir	8	21	29
Fechar	16	42	58
Remover	16	42	58
Renomear	64	186	250
Ler	2	3	5
Escrever	2	3	5
Ajuste do Ponteiro do Arquivo	1	1	2
Total de Cenários de Simulação			407

Tabela 1: Cenários de Simulação do sistema descentralizado

4.11.3 Testes das Operações de Leitura e Escrita

Os testes das operações de E/S buscaram certificar o correto funcionamento das funções do processo Cliente, que são responsáveis pela determinação dos Servidores envolvidos numa operação de leitura ou escrita e pela distribuição de dados a cada um destes. Os estados e as trocas de mensagens envolvidos, para a realização dessas operações, são idênticas nos dois tipos de sistema analisados. Isto porque os Clientes acessam diretamente os Servidores para ler ou escrever nos arquivos, independentemente da arquitetura possuir ou não um processo que centralize as demais funções.

Para a operação de escrita o Cliente deve determinar em qual *segmento* e *stripe* está localizada a posição inicial de armazenamento de dados. Se o volume de dados exceder o tamanho de uma unidade de distribuição, os dados a serem escritos são particionados em diversas unidades, determinando-se, em seguida, a qual segmento cada uma destas pertence. Havendo mais de uma unidade relacionada a um mesmo segmento, estas são agrupadas num mesmo bloco antes de serem enviadas ao Servidor responsável pelo segmento, minimizando assim as transferências de dados pela rede.

Na operação de leitura o Cliente define o segmento onde está o início do bloco de dados solicitado. Se o tamanho do bloco exceder ao de uma unidade de distribuição, serão determinados os outros segmentos envolvidos e, em seguida, serão contatados os Servidores correspondentes.

Para avaliar todas essas funcionalidades foi considerado um conjunto de situações que derivam diretamente das fórmulas que calculam a localização de um bloco de dados, as quais por sua vez levam em consideração o ponteiro do arquivo global (*offset*), o tamanho da unidade de distribuição utilizada (*str_unit*) e o número de segmentos em que o arquivo é distribuído (*nro_segmentos*). Tais fórmulas são definidas como (GUARDIA, 1999):

Stripe = offset / str_unit;
Segmento = stripe *mod* nro_segmentos;
Offset_segmento = (stripe / nro_segmentos) * str_unit + (offset *mod* str_unit);

Inicialmente são determinados o *stripe* e o *segmento* do arquivo distribuído, onde ocorrerá a operação de escrita ou de leitura. Em seguida o *Offset_segmento* calcula a posição inicial da operação dentro do segmento.

A partir dessas fórmulas diferentes configurações de distribuição de dados foram consideradas. Nos testes o volume de dados empregado forçava o Cliente a utilizar todos ou a maioria dos Servidores disponíveis. Os seguintes aspectos forma avaliados:

1) *Posição de início da leitura/escrita de dados:*

Se o *offset* = 0 (Figura 27a) a operação de leitura ou escrita ocorre a partir da posição inicial do arquivo ou de seu primeiro segmento. Para um *offset* ≠ 0 há duas possibilidades:

- se o valor do *offset* é um múltiplo do tamanho da unidade de distribuição (*offset mod str_unit = 0*), o início da operação coincide com o início da unidade de distribuição num dos segmentos que compõem o arquivo distribuído (Figura 27b);

- caso contrário (*offset mod st_unit ≠ 0*), o início da operação ocorre dentro de uma unidade de distribuição, o que implica num *offset* dentro da *str_unit* (Figura 27c).

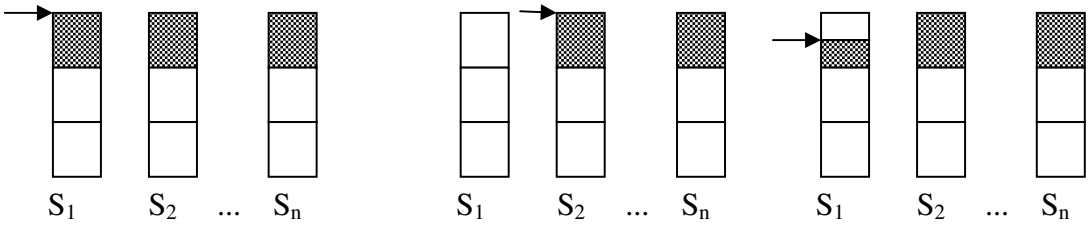


Figura 27a **Figura 27b** **Figura 27c**

Figura 27: Posição Inicial de E/S de Dados

2) Número de Stripes:

O número de *stripes* presentes numa operação depende da combinação de valores das variáveis envolvidas, o que pode gerar uma *stripe* incompleta (Figura 28a), completa (Figura 28b) ou mais de uma *stripe* (Figura 28c).

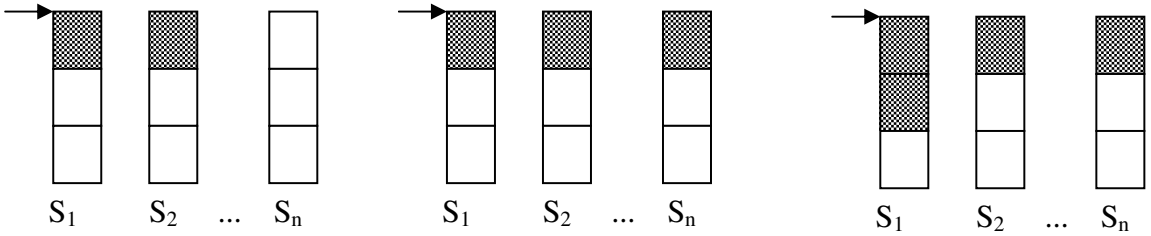


Figura 28a **Figura 28b** **Figura 28c**

Figura 28: Número de Stripes

3) Posição final de leitura e escrita

Diz respeito à nova posição do ponteiro do arquivo após a realização da operação. Há duas possibilidades:

- o ponteiro aponta para o fim de uma *str_unit* (Figura 29a);
- o ponteiro aponta para um ponto interno a uma *str_unit* (Figura 29b).

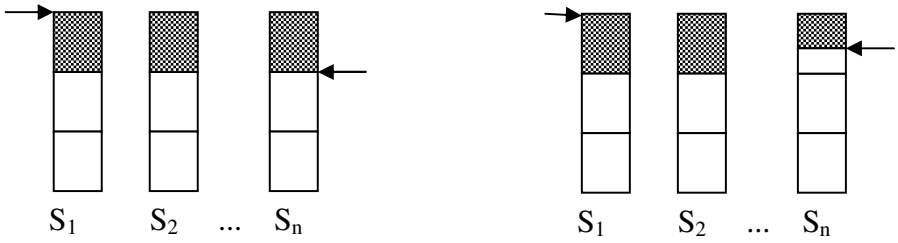


Figura 29a **Figura 29b**

Figura 29: Posição Final de E/S de Dados

Essas informações são relevantes para a distribuição dos blocos de arquivo numa operação de escrita e na remontagem desses blocos numa posterior leitura dos fragmentos.

4) Número de Servidores envolvidos

Conforme a quantidade de dados a serem lidos ou escritos, o Cliente deve utilizar pelo menos um Servidor, ou até todos os Servidores relacionados ao arquivo distribuído. Nos testes foram avaliadas as possíveis combinações de Servidores envolvidos com um mesmo arquivo.

Com a ajuda do *Edb* foram criadas macros para realizar, de forma automática, simulações que descrevem cada uma das possíveis combinações de situações. Basicamente as macros pré-configuram valores de *offset* e de volume de dados sobre um arquivo, que é previamente aberto com uma *str_unit* adequada às características dos serviços de E/S que se pretendia testar.

Para esquematizar os testes de leitura e escrita, considerou-se que um conjunto de testes seria aplicado sobre um determinado conjunto de Servidores, cada um destes responsável por um dos segmentos do arquivo. Em seguida as possíveis posições de início e término da operação, o volume de dados manipulado, que influencia na determinação dos Servidores disponíveis envolvidos com a operação, e o número de *stripes* foram considerados. Por exemplo, para um teste envolvendo quatro Servidores foram contempladas as situações apresentadas na Figura 30.

Início da Operação	Número de Segmentos	Término da Operação
1. Seg1 (offset = 0)	1. (nro_seg < = 4)	1. Seg1 (str_unit completa)
2. Seg2 (offset = str_unit)	2. (nro_seg > 4)	2. Seg2 (str_unit completa)
3. Seg3 (offset = 2*str_unit)		3. Seg3 (str_unit completa)
4. Seg4 (offset = 3*str_unit)		4. Seg4 (str_unit completa)
5. Seg1 (offset = n) (0 < n < str_unit)		5. Seg1 (str_unit incompleta)
6. Seg2 (offset = n*2)		6. Seg2 (str_unit incompleta)
7. Seg3 (offset = n*3)		7. Seg3 (str_unit incompleta)
8. Seg4 (offset = n*4)		8. Seg4 (str_unit incompleta)

Figura 30: Situações para as operações de leitura e escrita

A primeira coluna indica a posição inicial da operação. Nas situações de 1 a 4 a operação ocorre, respectivamente, no início de cada um dos quatro segmentos que compõem o arquivo (Figuras 27a e 27b). As situações de 5 a 8 também dizem respeito aos segmentos de 1 a 4, com a diferença que o início da operação não é mais coincidente com o início da unidade de distribuição (Figura 27c). Nesse caso um valor *n*, determinado entre zero e o tamanho da *str_unit* empregada, é atribuído ao *offset* do arquivo. Uma vez determinado o valor de *n*, calculado para o primeiro segmento (situação 5), este é mantido nos testes dos segmentos posteriores.

A segunda coluna indica o número de segmentos (*nro_seg*) envolvidos, estabelecidos nos testes pelo volume de dados considerado. Essa quantia cria as situações ilustradas na Figura 28 em relação ao número de segmentos.

A terceira coluna da tabela mostra as possíveis posições onde podem ocorrer o fim das operações de E/S. As situações de 1 a 4 descrevem os casos onde o final da operação coincide com o fim de uma unidade de distribuição, localizada nos segmentos de 1 a 4 respectivamente (Figura 29a). Nas situações de 5 a 8, que ocorrem correspondentemente aos segmentos de 1 a 4, o fim da operação não é coincidente com o final de uma unidade de distribuição (Figura 29b).

A combinação de todas as situações descritas nas três colunas leva à formação de 128 cenários de simulação, descritos através de macros que configuram os parâmetros das primitivas *pread* e *pwrite*, as quais são enviadas pelo Usuário ao sistema conforme a situação que se pretende avaliar.

Durante a execução de um teste o processo Cliente determina os segmentos envolvidos com a operação solicitada e, para cada segmento, determina a posição inicial onde será realizada a operação (*offset_segmento*) e a quantidade de dados a ser lida ou escrita (*count*). Esses valores são armazenados num arquivo pela macro que conduz o teste, podendo ser posteriormente verificados. Através de comandos de visualização, a troca de mensagens entre Cliente e Servidores é monitorada, possibilitando verificar se as mensagens estão sendo encaminhadas para os Servidores apropriados. Na leitura os dados retornados pelos Servidores podem ser conferidos e, durante a escrita, a correta distribuição de dados pode ser atestada.

Para a realização dos testes descritos nesta seção, foram implementas as funções do processo Servidor, que efetuam as operações sobre os discos (e.g., criação/abertura de arquivo, fechamento, leitura e escrita), fazendo com que os segmentos realmente fossem armazenados como arquivos, tornando assim os testes mais legítimos na medida em que possibilita a verificação do conteúdo desses arquivos.

Entretanto com a especificação ainda centralizada, a simulação de diversos Servidores, escrevendo cada um em seu próprio disco local, só pôde ser realizada atribuindo-se a cada Servidor um sub-diretório diferente do disco compartilhado. Dessa forma o arquivo distribuído é representado por diversos arquivos, com o mesmo nome, distribuídos em sub-diretórios diferentes de um mesmo disco.

5 Segundo Estudo de Caso: Network Parallel File System (NPFS)

Nos capítulos anteriores foi criado um modelo básico para a descrição formal de um sistema de arquivos descentralizado. Nesta seção a arquitetura proposta, para o sistema básico, será alterada para que um novo componente, denominado *Coordenador de Serviços* ou *Mestre*, possa ser incluído. Esse componente irá representar o processo que centralizará algumas operações sobre os arquivos distribuídos e que se responsabilizará pela administração dos meta-dados. As operações oferecidas aos Clientes do sistema, para a manipulação de arquivos distribuídos, serão preservadas no sistema a ser especificado. Exemplos de sistemas que apresentam essa arquitetura centralizada, descrita no Capítulo 1, são o NPFS e o PIOUS.

O uso de um processo Mestre simplifica as funções desempenhadas pelos processos Cliente e Servidor, já que este assume algumas das responsabilidades. Entretanto a sua presença no sistema leva a alterações nas MEFEs desses processos e à necessidade da concepção de uma nova MEFE para representar seu comportamento.

Essa seção demonstrará a elaboração de uma nova arquitetura, tendo como base a arquitetura do sistema descentralizado. Serão apresentadas as modificações necessárias sobre os processos Cliente e Servidor, permitindo determinar o grau de alteração no sistema básico provocado pela inclusão desse novo processo.

A arquitetura de um sistema de arquivos distribuídos centralizado, descrita na Figura 31, é baseada no estudo dos sistemas de arquivos distribuídos que possuem um processo Coordenador de Serviços. Esse processo Mestre, como será chamado, comunica-se com todos os processos Cliente e Servidor.

Dessa forma, o sistema centralizado passa a ser definido por três processos distintos:

- *Cliente* é um processo do Usuário, que utiliza um conjunto de primitivas disponíveis para acessar os dados compartilhados e que, dependendo do tipo de operação a ser realizada, comunica-se com o Mestre ou diretamente com os Servidores;

- *Mestre* é responsável pela iniciação do sistema, cuidando da ativação e manutenção dos Servidores. Este também gerencia algumas operações centralizadas, tais como a abertura e o fechamento de arquivos e as operações de controle compartilhado;

- *Servidor*, assim como no sistema descentralizado, é responsável pelo acesso aos fragmentos de arquivos, que estão armazenados localmente e se encontram nos computadores que compartilham seu disco local.

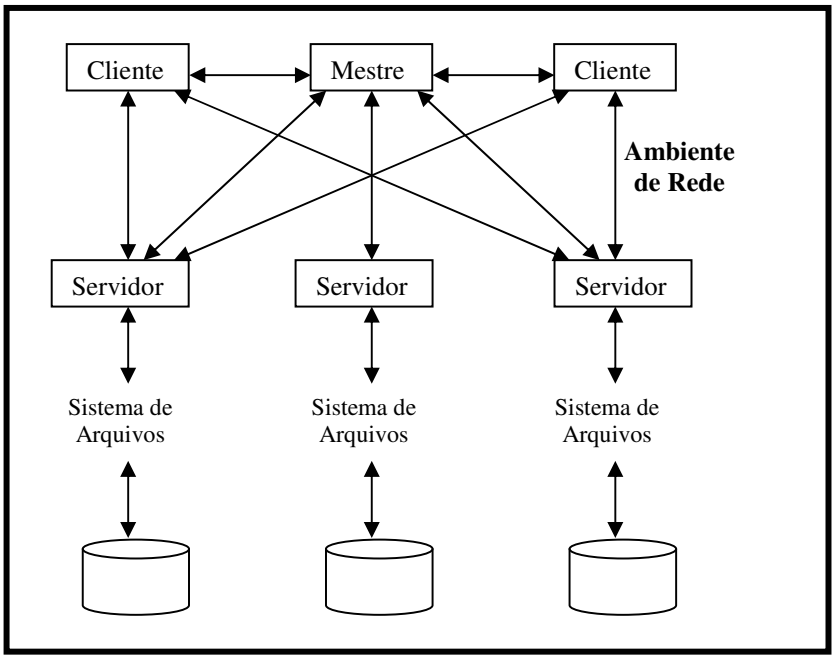


Figura 31: Arquitetura de um sistema de arquivos paralelos centralizado

As interações, que ocorrem entre os processos para a realização dos serviços, serão descritas nas próximas seções.

5.1 Protocolo de Comunicação

A ativação do sistema tem início com a iniciação do processo Mestre que, por sua vez, encarrega-se da ativação de um processo Servidor em cada uma das estações de

trabalho especificadas (os Servidores podem ser previamente determinados ou podem ser ativados e desativados dinamicamente).

Concluído esse passo inicial, qualquer Cliente pode solicitar ao Mestre a abertura de um arquivo distribuído. Se o arquivo for aberto com sucesso, o Cliente estará apto a executar, dentre outras operações, a leitura e a escrita nesse arquivo. Assim como no sistema básico, essas operações são realizadas pelos Clientes sem a intermediação do Mestre. Entretanto operações centralizadas, tais como remoção, renomeação e fechamento de arquivos, exigem a participação do processo Mestre. A Figura 32 ilustra as interações entre os componentes do sistema na realização de uma operação de abertura de um arquivo distribuído.

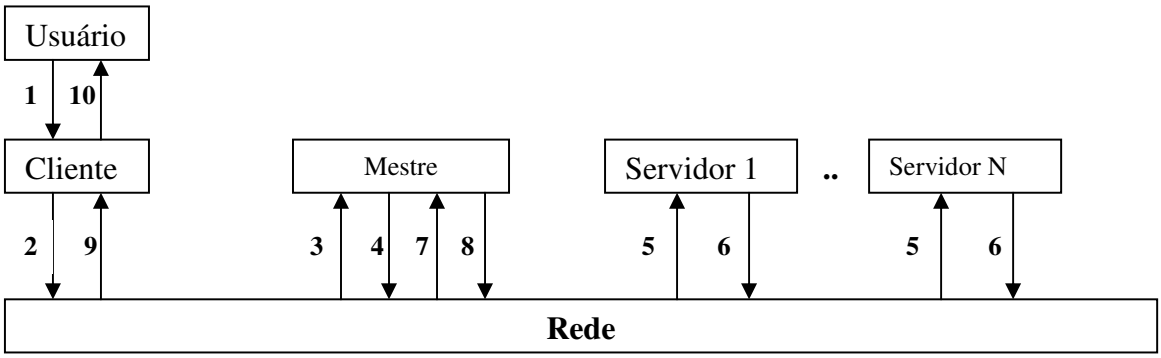


Figura 32: Interações entre processos durante a abertura de arquivo

A solicitação parte do Usuário (1), intermediado pelo seu processo Cliente (2). Assim que recebe uma mensagem para abertura de arquivo (3), o processo Mestre verifica a existência do mesmo, consultando uma estrutura de dados interna, onde são armazenadas informações sobre todos os arquivos distribuídos do sistema (meta-dados). Se o arquivo existir, o Mestre contacta todos os Servidores envolvidos com o arquivo (4, 5). Os Servidores, por sua vez, tentam abrir seus fragmentos locais e respondem ao Mestre quanto à solicitação (6). O Mestre recebe a resposta de todos os Servidores (7) e retorna ao Cliente solicitante o resultado da operação (8, 9), que pode ser de sucesso (todos os Servidores conseguiram abrir seus fragmentos locais) ou de erro (um ou mais Servidores não retornou ao Mestre com sucesso). Todo o procedimento é transparente para o Usuário, que recebe o resultado final (10) como se estivesse interagindo com um sistema centralizado de arquivos.

Se o arquivo a ser aberto pelo Cliente não existir, este poderá ser criado. Para criar um arquivo distribuído o Mestre precisa de um conjunto de Servidores, onde serão armazenados os segmentos desse arquivo. Esse conjunto pode ser fornecido pelo Cliente na mesma mensagem de solicitação de abertura do arquivo. Se o Cliente não fornecer essa informação, o Mestre ficará responsável pela definição de um conjunto de Servidores disponíveis.

A interação entre Cliente, Mestre e Servidores ocorre da mesma maneira tanto para a abertura quanto para o fechamento, renomeação e remoção de arquivos. A leitura e escrita de arquivos são realizadas diretamente entre o Cliente e os Servidores envolvidos, exatamente da mesma forma que são realizadas no sistema descentralizado, impedindo que o processo Mestre se torne um gargalo para o sistema. Algumas operações, tais como ocorre no sistema descentralizado, podem ser realizadas de forma síncrona ou assíncrona.

Nos sistemas centralizados é empregado um protocolo de comunicação não orientado a conexão, implicando que o Mestre, os Clientes e os Servidores devem possuir um mecanismo para recuperação de falhas na rede.

5.2 Primitivas de Serviço

O conjunto de primitivas para o sistema centralizado é o mesmo disponibilizado no sistema descentralizado. O prefixo *p*, para indicar operação paralela sobre os arquivos, é adicionado aos nomes das primitivas NPFS. A diferença na realização dos serviços das primitivas *p_open*, *p_close*, *p_unlink* e *p_rename*, em relação ao sistema especificado anteriormente, reside apenas na comunicação interna entre os processos, que envolve também o processo Mestre. Porém do ponto de vista dos Usuários, a operação do NPFS assemelha-se à do *Galley*.

A primitiva *lseek* atua diretamente sobre o Cliente, não havendo diferença em seu modo de atuação se comparado ao sistema descentralizado. O sistema centralizado também possibilita que os serviços relacionados às primitivas *p_unlink*, *p_rename* e *p_close* sejam realizados de forma síncrona ou assíncrona.

5.3 MEFEs dos Processos

Para a centralização dos serviços do sistema foi necessário criar uma MEFE, que determinasse o comportamento do novo componente Mestre, e realizar modificações na MEFE do processo Cliente. Essa MEFE foi criada para o sistema NPFS e empregada aqui como exemplo para um modelo genérico de um sistema centralizado.

Para o processo Mestre foi criada a MEFE descrita na Figura 33. Basicamente há dois estados principais: *Espera*, que representa um arquivo fechado sobre o qual podem ser realizadas operações de renomeação e remoção de arquivo, e *Aberto*, que libera as operações de escrita e leitura de dados aos Clientes e permite o recebimento de um pedido para o fechamento de arquivo distribuído.

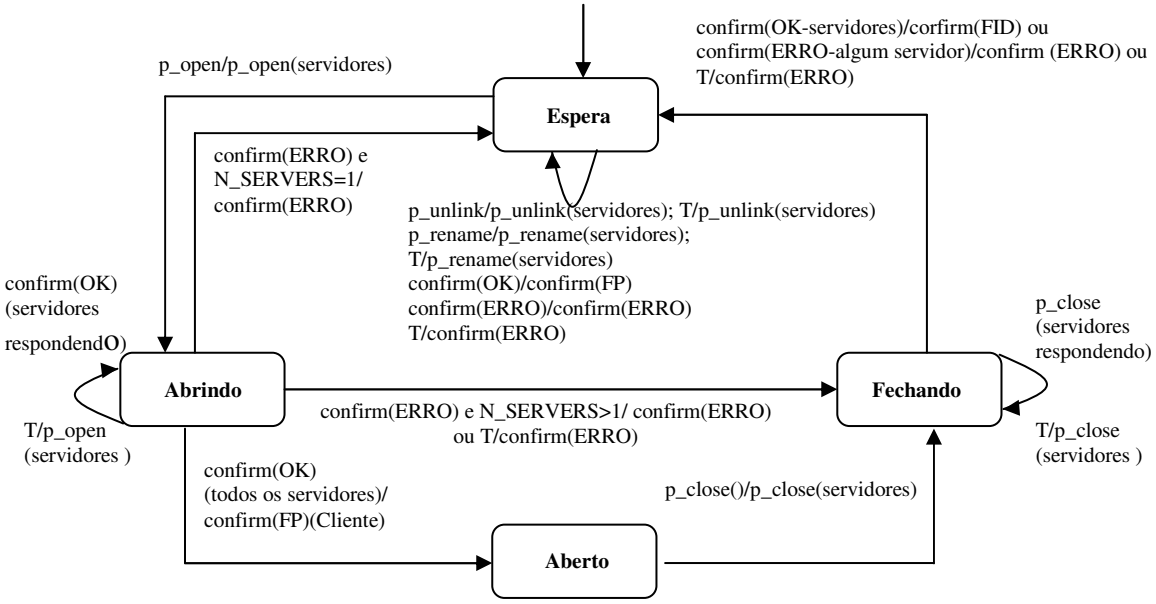


Figura 33: MEFE para o processo Mestre

O Mestre é responsável por contactar todos os Servidores, ao receber uma solicitação para remover ou renomear arquivos. Havendo ou não sucesso na operação, a MEFE retorna ao seu estado inicial e o resultado da operação é remetido ao Cliente que a solicitou.

Um arquivo somente pode ser considerado aberto, depois que todos os Servidores envolvidos abrirem com sucesso seus respectivos fragmentos locais. O estado *Abrindo* se incumbe dessa tarefa. Em caso de falhas na Rede ou nos Servidores, a MEFE responde com erro à solicitação do Cliente e retorna ao seu estado inicial (caso apenas um Servidor esteja relacionado ao arquivo sendo aberto), ou alcança o estado *Fechando*, a partir do qual é solicitado a todos os Servidores, que haviam respondido com sucesso à operação de abertura, para que fechem seus arquivos locais, anulando assim a operação de abertura. Esse estado também é responsável por fechar um arquivo aberto com sucesso. Após a operação, a MEFE retorna ao estado *Espera*.

A Figura 34 demonstra a nova MEFE para o processo Cliente, numa arquitetura onde há um processo centralizador de operações. Esta preserva o mesmo número de estados da MEFE originalmente concebida para um sistema descentralizado (Figura 20) e que foi utilizada como modelo.

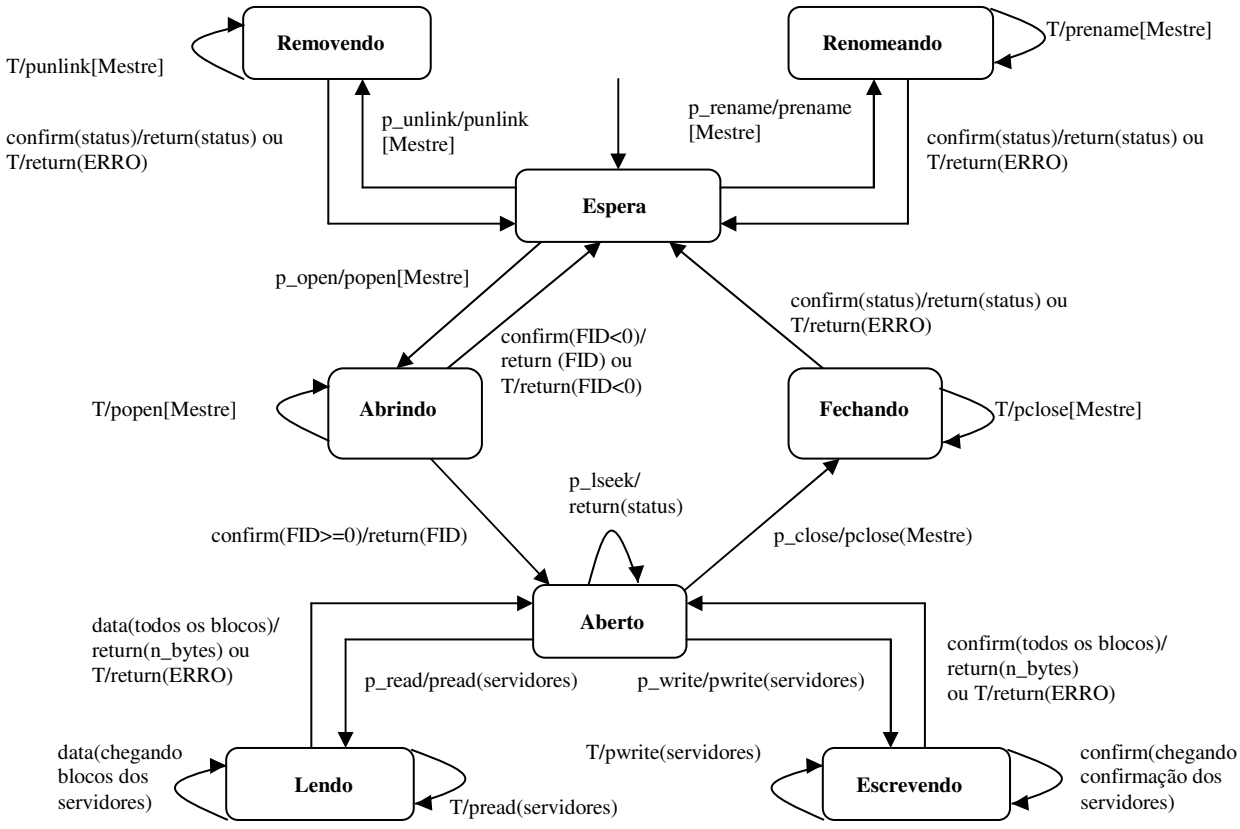


Figura 34: MEFE para o processo Cliente

A principal diferença é que os Clientes não precisam mais acessar diretamente os Servidores para realizar operações de renomeação, remoção e fechamento de arquivos. Essa tarefa ficou a cargo do processo Mestre. Isso levou a uma simplificação da MEFE original, já que todas as mensagens para realizar essas operações são agora enviadas ou recebidas apenas pelo processo Mestre, facilitando assim o controle de recebimento e retransmissão de mensagens. A MEFE permaneceu idêntica à original para os estados *Lendo* e *Escrevendo*.

A MEFE para o processo Servidor permanece a mesma apresentada na Figura 21 da seção 4.4. Seu comportamento, como provedor de acesso local aos fragmentos de arquivos distribuídos, não sofreu alterações. A diferença será a origem das solicitações de pedido de acesso, antes realizadas pelos Clientes e agora pelo Mestre, e o destino da resposta.

O uso do Mestre, como gerenciador de meta-dados, dispensa os Clientes de consultar essas informações, ao serem realizadas operações sobre os arquivos distribuídos. O próprio Mestre, ao receber uma solicitação, consulta os meta-dados armazenados internamente e verifica se a operação respeita a semântica de grupo, impedindo, por exemplo, que um Cliente tente remover um arquivo aberto por outro Cliente. A operação de renomeação foi simplificada consideravelmente, pois não existe mais a necessidade de migração de meta-dados entre processos Servidores.

5.4 Especificação formal do Sistema Centralizado em Estelle

A arquitetura Estelle do Sistema Centralizado, representada na Figura 35, preserva todos os módulos da arquitetura básica (*Cliente*, *Servidor*, *Rede*, *Usuário* e *Gerente*) e seus *links* de comunicação, e inclui um módulo extra para representar o processo *Mestre*.

Para os Usuários o sistema possui um comportamento idêntico ao sistema descentralizado, uma vez que a centralização de operações pelo processo Mestre é transparente e sua interferência apenas afeta a atuação dos processos internos.

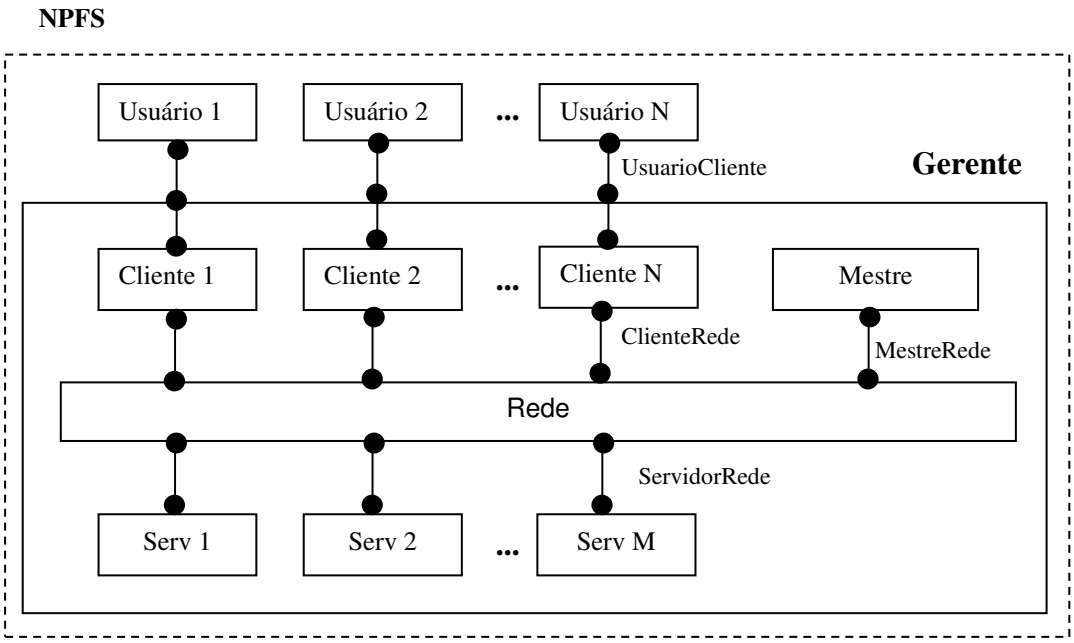


Figura 35: Arquitetura Estelle de um sistema de arquivos centralizado

Na comparação com a especificação equivalente do sistema descentralizado (Figura 13), foram incluídos a definição de um novo canal de comunicação e a declaração do módulo Mestre.

O canal *MestreRede* permite ao Mestre comunicar-se com Clientes e Servidores através do módulo de Rede, que recebeu um ponto de interação adicional com a finalidade de estabelecer um novo *link* por meio desse canal. Os demais módulos não sofreram nenhuma alteração em relação a seus cabeçalhos e ao modo como estão interligados.

As primitivas definidas nos canais de comunicação também passaram por alterações. Foram excluídas as primitivas responsáveis pelo acesso aos meta-dados nos Meta-servidores, funcionalidade que não está presente nesse novo sistema. Primitivas foram incluídas ou modificadas para a comunicação com o Mestre.

```
specification Sistema;

default individual queue;
timescale seconds;

{ parte de declaração do Módulo NPFS }
const
  N_usuarios =  any integer; {nro de Usuários do Sistema}
  N_servidores =  any integer;  {nro de Servidores do
Sistema }
 Cli = any integer;    { numero maximo de arquivos que
um cliente pode manipular simultaneamente }
  Serv = any integer; { numero maximo de arquivos que
um servidor pode manipular simultaneamente }
  Mtr = any integer;   { numero maximo de arquivos que o
Mestre pode manipular simultaneamente }..
  ...
{ declaração dos canais de comunicação }
channel UsuarioCliente(usuario,provedor);     ... ;
channel ClienteRede(usuario,provedor);   ... ;
channel MestreRede(usuario,provedor);    ... ;
channel ServidorRede(usuario,provedor);   ... ;

{ definicao do Usuário ----------------------------- }
module Usuario systemprocess; { cabeçalho do Usuário }
  ip U: UsuarioCliente(usuario);   { p. i. externo }
end;

body CorpoUsuario for Usuario;   { corpo do Usuário }
external;

{ definicao do Modulo Gerente -------------------- }
module Gerente systemprocess; { cabeçalho do Gerente }
 {pontos de interação (p. i.) externos }
 ip UC: array[1..N_usuarios] of UsuarioCliente(provedor)
end;

body CorpoGerente for Gerente; { corpo do Gerente }
  { declaração do Módulo Cliente --------------- }
  module Cliente process (ind_cli:integer);  { cabeçalho }
    ip C: UsuarioCliente(provedor);  { p. i. externos }
       R: ClienteRede(usuario);
  end;

  body CorpoCliente for Cliente;  { corpo do Cliente }
    ...
  end; { fim da especificação do corpo do Cliente }

 { declaração do Módulo Mestre ----------------  }
 module Mestre process;   { cabeçalho do Mestre }
   ip MS: MestreRede(provedor);  { p. i. externo }
 end;

 body CorpoMestre for Mestre;    { corpo do Mestre }
   ...
 end; { Fim da especificacao do corpo do Mestre }

{ declaração do Módulo Servidor ---------------     }
module Servidor process (ind_serv:integer);{ cabeçalho }
  ip S: ServidorRede(provedor);   { p. i. externo }
...
end; {Fim da especificação do corpo do Servidor }

{ declaração do Módulo de Rede ----------------------}
module Rede process;       { cabeçalho da Rede }
{ pontos de interação externos }
ip RC:array[1..N_usuarios] of ClienteRede(provedor);
   MR:MestreRede(usuario);
   RS:array[1..N_Servidores] of ServidorRede(usuario);
end;

body CorpoRede for Rede;   { corpo da Rede }
 ...
end; { Fim da especificação do corpo da Rede }

{ parte de declaração do Módulo Gerente ---------------- }
{ declaração de variáveis de módulo}
  modvar MCliente:array[1..N_usuarios] of Cliente;
       MMestre:Mestre;
       MServidor:array[1..N_servidores] of Servidor;
       MRede:Rede;
  initialize      { criação estática das ocorrência  }
  begin
    init MMestre with CorpoMestre; {  Módulo Mestre }
    init MRede with CorpoRede;      { Módulo de Rede }
    all i:1..N_servidores do
    begin    { Módulos Servidores }
         init MServidor[i] with CorpoServidor(i);
              { conexão dos Servidores com a Rede }
         connect MServidor[i].S to MRede.RS[i]
    end;
    all i:1..N_usuarios do
    begin      { criação dos Módulos Cliente }
        init MCliente[i] with CorpoCliente(i);
              { conexão dos Clientes com a Rede }
         connect MCliente[i].R to MRede.RC[i];
              { vinculação dos Clientes com seus
 respectivos Usuários }
        attach UC[i] to MCliente[i].C;
    end;
    { conexão do Mestre com a Rede }
     connect MRede.MR to MMestre.MS;
  end;
 end; { Fim do Gerente }

{ parte de declaração do Módulo NPFS ------------------- }
{ declaração de variáveis de Módulo }
 modvar MUsuario:array[1..N_usuarios] of Usuario;
          MGerente:Gerente;
 initialize  {criação das ocorrências estáticas do Usuário}
 begin
   { criação do Módulo Gerente }
   init MGerente with CorpoGerente;
   all i:1..N_usuarios do
   begin
        { criação dos Módulos Usuários }
       init MUsuario[i] with CorpoUsuario;
       { conexão dos Usuários com o Gerente }
      connect MUsuario[i].U to MGerente.UC[i];
   end;
 end;
```

Figura 36: Esqueleto da especificação centralizada

5.4.1 Refinamento da Especificação

No refinamento do sistema centralizado é aplicada a mesma estratégia do sistema derivado do modelo básico, na qual a criação e remoção dinâmica de instâncias de módulos refletem a capacidade do sistema em manipular simultaneamente diversos arquivos. Os módulos Cliente e Servidor permanecem sem alteração nesse nível de abstração da arquitetura do sistema. O módulo Mestre, que também tem que ser capaz de controlar diversos arquivos, é refinado de maneira semelhante aos demais módulos. A Figura 37 demonstra o refinamento da arquitetura inicial para os principais processos do Sistema.

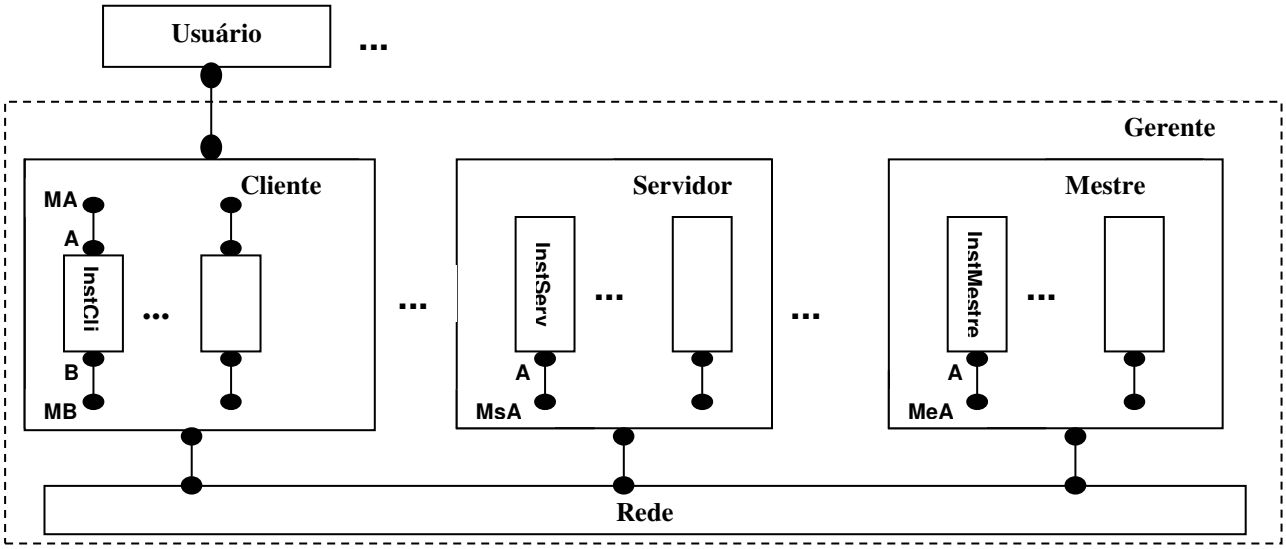


Figura 37: Refinamentos dos módulos Cliente, Servidor e Mestre

Os esqueletos das especificações dos módulos Cliente e Servidor são os mesmos do sistema básico descentralizado, respectivamente apresentados nas Figuras 40 e 41. A funcionalidade e especificação do módulo Mestre são apresentadas na seção subseqüente.

5.4.2 O Módulo Mestre

O Mestre cria uma nova instância filha *InstMestre* e estabelece seus *links* de comunicação quando o módulo Rede solicita a abertura, remoção ou renomeação de um arquivo. Uma *InstMestre* e seus *links* são removidos quando o arquivo, que está sob sua responsabilidade, é fechado ou quando uma operação de remoção ou renomeação é encerrada.

A Figura 38 ilustra o esqueleto da especificação do corpo de Mestre, contendo a definição de seu módulo filho, sendo que o comportamento de *InstMestre* reflete a MEFE apresentada na Figura 33.

```
body CorpoMestre for Mestre;
  ip MeA: array[1..Mtr] of MestreRede(usuario); { pontos de interacao interno }

 { definicao do modulo filho do Mestre}
  module InstMestre process (inst_mtr:integer);
     ip A: MestreRede(provedor);   { ponto de interação externo }
     export Fim:boolean;           { variáveis exportadas }
            Ind_Inst:integer;
  end;

   body CorpoInstMestre for InstMestre;
     {definição da MEFE do módulo filho do Mestre}
   end; {CorpoInstMestre }
     ...
   trans
     { criação dinâmica de módulos }
     ...
end; { CorpoMestre }
```

Figura 38: Esqueleto da especificação do corpo de Mestre

A comunicação entre o Mestre e suas instâncias filhas é realizada através da troca de mensagens, semelhantemente ao que ocorre no módulo Cliente, e também através de variáveis exportadas pelas *InstMestre*. Essas variáveis são lidas pelo Mestre para verificar e remover as instâncias que já consumiram serviços.

Os meta-dados são armazenados em estruturas de dados e são consultados pelo Mestre ao ser solicitada uma operação de abertura, fechamento, renomeação ou remoção de arquivo.

5.5 Especificação formal segundo o framework conceitual

A especificação formal do sistema segue as etapas descritas para a especificação do sistema descentralizado. Deverão ser descritas em Estelle as MEFEs, correspondentes aos comportamentos dos módulos filhos dos processos Cliente, Mestre e Servidor, que são responsáveis pelo gerenciamento dos arquivos distribuídos. Os comportamentos dos módulos pai são definidos por transições que controlam o envio e recebimento de primitivas entre as instâncias de seus módulos filhos e a rede. Essas transições também cuidam da criação dinâmica de instâncias de módulos filhos e monitoram a solicitação de remoção dessas instâncias, que pode ocorrer em alguns casos.

Funções e procedimentos, devotados ao controle e acesso de informações internas aos processos, podem ser declarados como primitivas, permitindo assim ao especificador escolher a melhor estrutura de dados para manipulá-las. Neste framework conceitual um conjunto básico de funções e procedimentos é oferecido conjuntamente com a especificação, sendo que este pode ser facilmente substituído por outro conjunto que cumpra o mesmo papel. A especificação completa do NPFS em Estelle está disponível em (MANTOVAN, 2000).

Como o sistema centralizado segue a mesma estratégia de especificação do descentralizado e compartilha algumas de suas características estruturais e comportamentais, sua especificação pode ser obtida mais rapidamente. A principal diferença entre os dois sistemas está na inclusão do módulo Mestre e na redefinição da MEFE do módulo Cliente.

O módulo Servidor possui o mesmo comportamento (MEFE) nos dois sistemas. Como diferença, suas funções que gerenciam os meta-dados na versão descentralizada são incorporadas pelo Mestre na especificação centralizada. A Rede continua desempenhando o mesmo papel, alterando-se apenas as interações que recebe e deve encaminhar.

No interior dos módulos Cliente e Servidor, as condições que definem a criação e remoção dinâmica de instâncias de submódulos, regidas pelas solicitações de serviços, são as mesmas tanto no *Galley* quanto no NPFS. Estes também empregam o mesmo método de distribuição e recuperação de dados dos fragmentos dos arquivos

distribuídos, bem como mecanismos de retransmissão de mensagens e funções que armazenam informações internas aos módulos.

Características em comum permitem o reaproveitamento de parte do código da especificação sem alteração ou à necessidade de se fazer apenas ajustes pontuais em determinadas partes desse código para que se possa obter um novo sistema a partir de uma outra especificação já disponível, processo que diminui o tempo de codificação do novo sistema. Uma análise entre o NPFS e *Galley* demonstra que esses sistemas apresentam até 60% de similaridade, total ou parcial, em suas especificações Estelle.

5.6 Simulação do Sistema Centralizado

A estratégia aplicada ao sistema descentralizado também foi aplicada para validar o centralizado. Em comparação ao sistema descentralizado, foi necessária a criação dos cenários que o processo Mestre assume durante a realização dos serviços. Por outro lado, os cenários do Cliente do novo sistema se tornaram mais simples que os cenários obtidos para o mesmo processo do sistema descentralizado. Isso se deve ao fato do Mestre intermediar algumas operações que os Clientes realizavam enviando e recebendo mensagens diretamente dos Servidores e Meta-servidores. Assim como realizado na seção 4.11.2, as operações foram divididas em dois conjuntos de cenários, conforme o sucesso ou falha da operação. A Figura 39 apresenta as situações para o caso de operação de abertura de arquivo bem-sucedida. A representação das situações segue aquela definida para o sistema descentralizado.

Para abrir um arquivo, conforme descrito na Situação_1, um Cliente envia agora uma solicitação *p_open* para o Mestre, que passa a ter a responsabilidade de atualizar os meta-dados e entrar em contato com os Servidores envolvidos. Dessa maneira o Cliente passa a ter, na versão centralizada, apenas dois cenários para abertura com sucesso de arquivo, os quais representam o acesso direto ao Mestre ou o acesso após uma ou mais tentativas de contato. O Mestre possui cenários análogos com relação ao Servidor, que sempre confirmará a abertura de arquivo.

Abrir Arquivo: operação com sucesso

Sit1_Cli: arquivo aberto sem que ocorra problemas
(ESPERA)p_open[U](ABRINDO)popen[M] → (ABRINDO)return_OK[M] (ABERTO)return_OK[U]
Sit2_Cli: Cliente retransmite pedido de abertura até conseguir resposta do Mestre
(ESPERA)p_open[U](ABRINDO)popen[M] → (ABRINDO)_(ABRINDO)popen[M] → ... →
(ABRINDO)_(ABRINDO)p_open[M] → (ABRINDO)return_OK[M] (ABERTO)return_OK[U]

Sit1_Me: Mestre abre arquivo sem problemas
(ESPERA)p_open[C](ABRINDO)p_open[S] → (ABRINDO)confirm_OK[S](ABERTO)confirm_OK[C]
Sit2_Me: Mestre retransmite pedido de abertura, aos servidores que não responderam, até conseguir resposta
(ESPERA)p_open[C](ABRINDO)p_open[S] → (ABRINDO)_(ABRINDO)p_open[S] → ... →
(ABRINDO)_(ABRINDO)p_open[S] → (ABRINDO)confirm_OK[S](ABERTO)confirm_OK[C]

Sit1_Serv: Servidor abre arquivo com sucesso
(ESPERA)p_open[M](ABERTO)return_OK[M]

Cenários que levam à abertura do arquivo com sucesso
Cenário1: (Sit1_Cli) → (Sit1_Me) → (Sit1_Serv)
Cenário2: (Sit1_Cli) → (Sit2_Me) → (Sit1_Serv)
Cenário3: (Sit2_Cli) → (Sit1_Me) → (Sit1_Serv)
Cenário4: (Sit2_Cli) → (Sit2_Me) → (Sit1_Serv)

Figura 39: Cenários da tentativa bem sucedida de abertura de arquivo

A inclusão de um novo processo diminuiu significativamente o número de mensagens trocadas entre os processos durante a abertura de um arquivo. Conseqüentemente, isso diminuiu o número de situações do Cliente, justificando assim um conjunto menor de cenários de simulação. O mesmo é verificado em relação aos cenários para as situações mal-sucedidas e também em relação às demais operações (principalmente a renomeação de arquivo).

As falhas que podem ocorrer durante a operação estão descritas na Figura 40. Em relação ao Cliente um erro pode ocorrer, por este não conseguir enviar a sua solicitação ao Mestre, ou devido à falha do Mestre ao tentar abrir o arquivo. O Mestre pode falhar devido a problemas na Rede, ou devido à recusa de um Servidor em abrir o arquivo local.

Abrir Arquivo: ocorre erro durante a operação

Sit1_Cli: Mestre retorna ERRO ao Cliente na primeira transmissão
(ESPERA)p_open[U](ABRINDO)popen[M] → (ABRINDO)return_ERRO[M] (ESPERA)return_ERRO[U]
Sit2_Cli: Cliente retransmite pedido até contactar Mestre, porém recebe ERRO como resposta
(ESPERA)p_open[U](ABRINDO)popen[M] → (ABRINDO)_(ABRINDO)popen[M] → ... →
(ABRINDO)_(ABRINDO)popen[M] → (ABRINDO)return_ERRO[M] (ESPERA)return_ERRO[U]
Sit3_Cli: Cliente retransmite pedido, mas não consegue contactar Mestre
(ESPERA)p_open[U](ABRINDO)popen[M] → (ABRINDO)_(ABRINDO)popen[M] → ... →
(ABRINDO)_(ABRINDO)popen[M] → (ABRINDO)_ (ESPERA)return_ERRO[U]

Sit1_Me: Mestre contacta Servidor envolvido, mas este retorna ERRO
(ESPERA)p_open[C](ABRINDO)p_open[S] → (ABRINDO)confirm_ERRO[S](ESPERA)confirm_ERRO[C]
Sit2_Me: Mestre tenta contato com Servidor após falha(s) na Rede. Ao conseguir, Servidor retorna ERRO
(ESPERA)p_open[C](ABRINDO)p_open[S] → (ABRINDO)_(ABRINDO)p_open[S] → ... →
(ABRINDO)_(ABRINDO)p_open[S] → (ABRINDO)confirm_ERRO[S](ESPERA)confirm_ERRO[C]
Sit3_Me: Mestre não consegue contactar Servidor envolvido após diversas tentativas de retransmissão
(ESPERA)p_open[C](ABRINDO)p_open[S] → (ABRINDO)_(ABRINDO)p_open[S] → ... →
(ABRINDO)_(ABRINDO)p_open[S] → (ABRINDO)_(ESPERA)confirm_ERRO[C]

Sit1_Serv: Servidor não consegue abrir o arquivo
(ESPERA)p_open[M](ESPERA)return_ERRO[M]

Cenários que levam à falha na abertura do arquivo
Cenário1: (Sit1_Cli) → (Sit1_Me) → (Sit1_Serv)
Cenário2: (Sit1_Cli) → (Sit2_Me) → (Sit1_Serv)
Cenário3: (Sit1_Cli) → (Sit3_Me)
Cenário4: (Sit2_Cli) → (Sit1_Me) → (Sit1_Serv)
Cenário5: (Sit2_Cli) → (Sit2_Me) → (Sit1_Serv)
Cenário6: (Sit2_Cli) → (Sit3_Me)
Cenário7: (Sit3_Cli)

Figura 40: Cenários da tentativa mal sucedida de abertura de arquivo

As operações de leitura e escrita são realizadas da mesma forma nos sistemas descentralizado e centralizado. Por essa razão, a estratégia de teste, descrita na seção 4.1.10.3, é aplicável sem alteração aos dois tipos de sistema.

A Tabela 2 mostra o número de cenários de simulação obtido para cada primitiva de serviço oferecida pelo NPFS. A descrição de todos os cenários está disponível em (MANTOVAN, 2000).

O menor número de cenários de simulação obtidos com o sistema centralizado em comparação com o que foi obtido com o sistema descentralizado, reflete o menor número de interações entre os processos que ocorre durante a realização das operações.

	Cenários de Simulação		
Operações	Bem Sucedidas	Mal Sucedidas	Total
Abrir	4	7	11
Fechar	8	14	22
Remover	8	14	22
Renomear	8	14	22
Ler	2	3	5
Escrever	2	3	5
Ajuste do Ponteiro do Arquivo	1	1	2
Total de Cenários de Simulação			89

Tabela 2: Cenários de Simulação do sistema centralizado

6 Implementação e Teste de Sistemas

Neste capítulo serão descritas as etapas para a implementação de um sistema de arquivos distribuídos a partir de sua especificação formal. O compilador Ec será utilizado para gerar a maior parte do código dos processos, mapeando estruturas Estelle em estruturas equivalentes na linguagem C.

Como estudo de caso foi escolhido o sistema centralizado NPFS, por possuir uma arquitetura mais complexa, com um maior número de processos. Existe também disponível uma versão implementada manualmente do NPFS que pode ser comparada, em termos de desempenho e quantidade de código gerado, com a versão do sistema gerada semi-automaticamente apresentada neste capítulo. A estratégia para a realização do sistema pode ser aplicada para o sistema base, sendo extensível para sistemas que incluem outros processos além do processo coordenador de serviços.

O compilador recebe como entrada uma especificação formal e cria, para cada módulo que compõe o sistema especificado, um conjunto de arquivos em código C independentes de plataforma de implementação. Todas as regras semânticas de Estelle, descritas no capítulo 4, seção 4.4.7, são respeitadas pelo compilador ao gerar os arquivos executáveis, sendo que estes possuem um gerenciador interno responsável por localizar e disparar as transições, que devem ser executadas paralelamente num determinado instante de tempo.

Não existe paralelismo real entre módulos Estelle, o que implica que recursos de implementação, que podem ser disponibilizados pelo ambiente, tais como processos ou *threads*, não são explorados. Seguindo a semântica dessa TDF somente uma transição pode ser executada de cada vez.

Após a atuação do compilador Ec ainda restarão lacunas presentes na especificação, que representam as partes do sistema dependentes do ambiente de execução, que deverão ser codificadas manualmente. Se mais de um mecanismo é oferecido (e.g., para a comunicação entre processos distribuídos), o implementador pode optar por aquele que melhor atende aos requisitos do sistema sendo implementado. Por exemplo para o sistema de arquivos distribuídos foi escolhido um mecanismo de comunicação que oferecia uma menor sobrecarga no transporte das primitivas, em detrimento de outro mecanismo que oferecia uma comunicação mais confiável. Dessa forma buscou-se aumentar o desempenho das operações de E/S, que é o principal objetivo desse tipo de sistema.

6.1 Implementação do sistema NPFS

A implementação semi-automática do Sistema Centralizado compreendeu dois passos. No primeiro a especificação inicial, que representava todo o sistema, foi dividida em três especificações independentes, cada uma correspondendo a um dos processos do sistema distribuído (Cliente, Mestre e Servidor). No segundo foram implementados os mecanismos de comunicação e as funções presentes nesses processos, que não foram definidas na especificação (declaradas em *Estelle* como *primitive*).

Para dividir a especificação do sistema, foi utilizada uma ferramenta do *EDT* denominada *Universal Generator (Ug)* (BULL, 2000c). Esta cria uma especificação separada para cada subsistema (módulos com atribuição *systemprocess* ou *systemactivity*) que é definido na especificação *Estelle*.

6.1.1 Modificações na Arquitetura Inicial

Na especificação Estelle original os módulos Usuário e Gerente são declarados como sistemas (*systemprocess*) e todos os demais como processos (*process*). Com essa estruturação o *Ug* não conseguiria dividir a especificação de maneira apropriada, pois geraria apenas uma especificação para o Usuário e uma outra para o Gerente, que por sua vez confinaria todos os processos distribuídos do *NPFS*. Portanto foi necessário adequar a especificação inicial à estratégia de divisão da ferramenta, a fim de que o *Ug* pudesse criar corretamente as três especificações pretendidas para os processos do Sistema de Arquivos.

Para a correta divisão da especificação os processos Cliente, Mestre e Servidor deveriam ser declarados como subsistemas (*systemprocess* ou *systemactivity*) do módulo principal que os engloba. Como esses módulos são filhos do módulo Gerente, as regras de estruturação de Estelle obrigavam que estes fossem declarados apenas como processos ou atividades (*process* ou *activity*). A solução foi remover o módulo Gerente da especificação inicial, já que este não interfere no comportamento do sistema, pois era usado apenas para ocultar dos Usuários os módulos constituintes desse sistema.

A segunda alteração na especificação inicial foi a do módulo Rede, o qual centralizava toda a comunicação interprocessos. Permanecendo como um módulo a parte, o *Ug* geraria um processo Rede, a ser executado num computador distinto dos que conteriam os demais processos que o utilizam. A solução foi descentralizar a comunicação, associando a cada módulo Cliente, Mestre e Servidor um módulo Rede específico, responsável unicamente pela comunicação do processo a este associado. O resultado final levou às estruturas representadas nas Figuras 41 e 42.

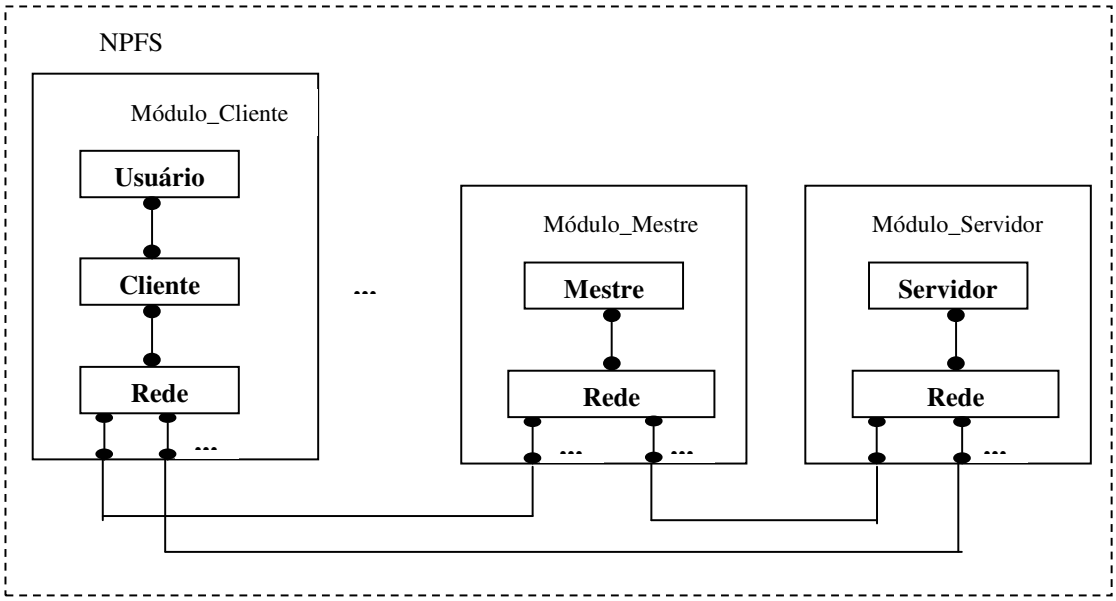


Figura 41: Modificações na arquitetura inicial

Após a divisão da especificação original, para que cada módulo Servidor pudesse ficar associado numa mesma especificação ao seu módulo Rede correspondente, tornou-se necessário criar um módulo subsistema mais amplo, que englobasse Servidor e Rede como módulos irmãos. Com esse objetivo foi criado o módulo *Modulo_Servidor*, que encapsula o Servidor junto com a sua Rede. Com o mesmo objetivo, em relação ao Mestre e ao Cliente, foram criados os módulos *Modulo_Mestre* e *Modulo_Cliente*, sendo que este último também encapsula o Usuário.

O desmembramento do módulo Rede exigiu a criação de um maior número de pontos de interação externos e de *links* entre esses pontos, diminuindo a abstração possibilitada pela existência de um único módulo Rede, o qual centralizava todas as funções de comunicação.

Decorrida essa etapa o sistema passou a ser constituído por três módulos subsistemas, que devem ser distribuídos em computadores diferentes para a execução em paralelo (Figura 42).

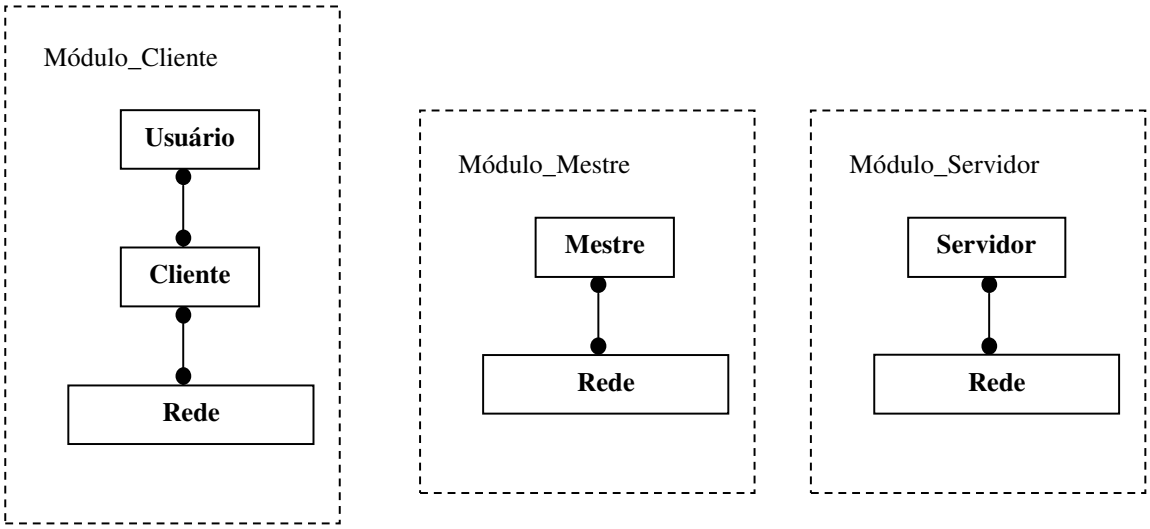


Figura 42: Divisão da especificação inicial

Os módulos Cliente, Mestre e Servidor, validados na especificação global, não sofreram nenhuma alteração em sua estrutura após a divisão da especificação.

Nos módulos Rede devem ser inseridas as funções que serão responsáveis pela comunicação (acesso à rede) do processo, as quais incluem a criação e iniciação de *sockets* e o envio e recebimento de mensagens.

Cada arquivo pode agora ser compilado com o *Ec* ou com um outro compilador *Estelle*, para que um arquivo executável possa ser gerado. Com a criação de funções para os procedimentos *primitive* da especificação inicial, e a utilização de funções voltadas à comunicação, os programas executáveis criados a partir do compilador *C* passam a ser específicos para uma determinada plataforma (Figura 43).

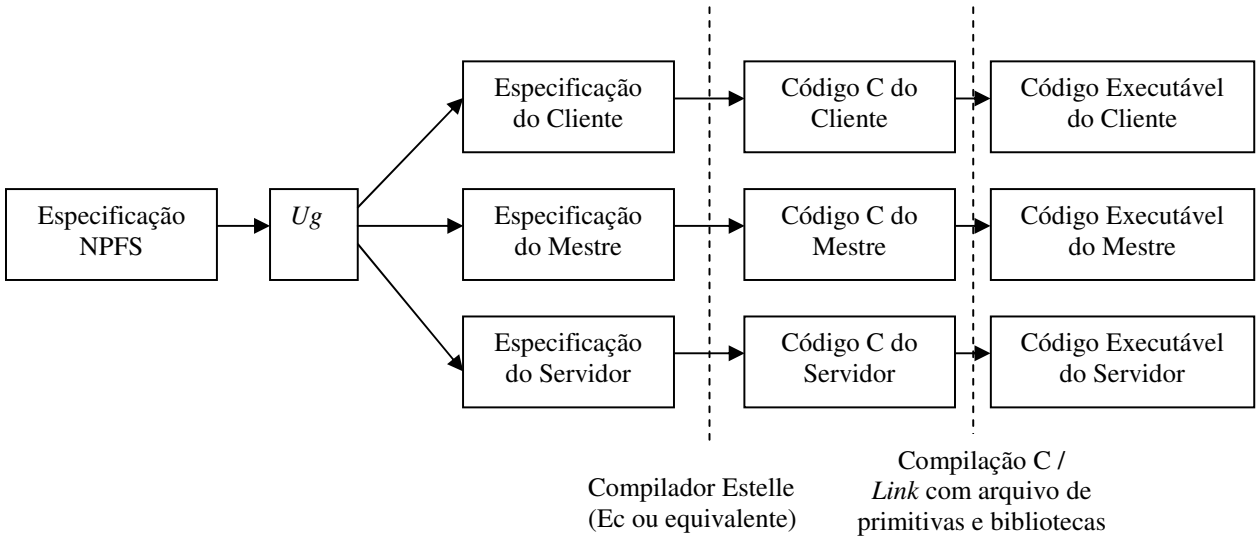


Figura 43: Etapas para a implementação de um sistema distribuído

Cada programa executável pode ser agora instalado num computador conectado em rede. Para Clientes e Servidores mais de uma instância do mesmo código pode ser instalada em máquinas diferentes.

A especificação completa do NPFS, submetida ao *Ug*, possuía cerca de 4700 linhas de código Estelle. No final do processo de divisão da especificação e compilação pelo *Ec*, foram gerados, somados os códigos dos processos Cliente, Mestre e Servidor, mais de 32000 linhas de código C. Dessa maneira, um programador, utilizando uma linguagem de especificação, codifica cerca de 15% do que seria necessário codificar caso empregasse diretamente uma linguagem de programação para implementar o sistema.

A relação de uma linha de código Estelle para aproximadamente sete linhas em C pode ser considerada ainda maior se for levado em conta que algumas funções de implementação comuns a todos os processos, responsáveis pelo controle da comunicação dos módulos e pelo mecanismo de escolha e execução de transições, entre outras, são disponibilizadas em bibliotecas ao invés de serem incorporadas ao código fonte dos processos.

6.1.2 O Módulo de Rede

Cada um dos processos do sistema possui um módulo Rede responsável por sua comunicação. Nesses módulos são implementadas primitivas de comunicação, que se encarregam da criação de *sockets* e do controle de transmissão e recepção de mensagens entre os processos. São quatro os tipos básicos de primitivas:

- *mxinit* faz a iniciação do mecanismo de *socket* assim que o processo é colocado em execução;

- *mxsend* envia uma interação para um processo remoto;

- *mxwhen* monitora um ou mais *sockets* a espera da chegada de interações de processos remotos;

- *mxoutput* recupera interações recém chegadas do *buffer* do socket e as envia, através de um ponto de interação interno, para o processo receptor local. Essa primitiva opera em conjunto com a primitiva *mxwhen*.

Para exemplificar o funcionamento da Rede será considerado o processo Cliente. Assim que esse processo é ativado, seu módulo Rede correspondente tenta iniciar (por meio da primitiva *mxinit* presente em sua parte de inicialização) um *socket*, usando como parâmetros o *hostname* da máquina, onde se encontra, e uma porta por *default* ou que é passada ao processo como um argumento.

Concluída essa etapa, a Rede fica aguardando que o Cliente solicite uma transmissão ou que algum processo transmita ao mesmo, sendo que essa situação é monitorada por *mxwhen*. Ao chegar uma mensagem, *mxwhen* ativa a função *mxoutput* que, por sua vez, extrai a transição do *buffer* do *socket* e a envia ao Cliente através de um ponto de interação interno.

Para enviar uma interação do Cliente a um processo remoto, a Rede utiliza a função *mxsend*, que empacota os parâmetros junto com um identificador de interação, o qual será usado pelo processo destino para reconhecer o transmissor. O funcionamento da Rede nos processos Mestre e Servidor é análogo ao do Cliente.

6.1.3 Complementação da Especificação

Na especificação Estelle algumas funções e procedimentos são declarados como sendo *primitive*, indicando que suas definições ficam a cargo do implementador. Por exemplo nos módulos Servidor as funções de abertura, fechamento, renomeação, remoção, leitura e escrita nos arquivos possuem apenas a sua assinatura, isto é, o nome da função acompanhado de seus parâmetros com seus respectivos tipos.

Para a implementação do sistema utilizando as ferramentas do *EDT*, todas essas funções precisam ser codificadas em *C* num arquivo a parte, o qual deve ser compilado e ligado (*linked*) ao código gerado pelo compilador *Ec*. A Figura 44 exemplifica como deve ser realizada a implementação de uma função. Os nomes da função e dos parâmetros, descritos em Pascal na especificação Estelle, precisam permanecer os mesmos e com letras minúsculas ao serem descritos

em *C* e os tipos dos parâmetros devem ser equivalentes nas duas linguagens. A implementação das primitivas incorpora características do ambiente onde serão executados os processos do sistema.

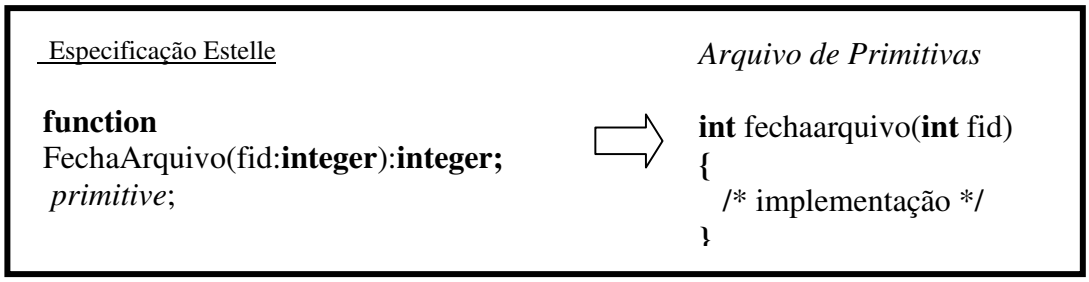


Figura 44: Implementação de função primitive

O código C escrito manualmente, que corresponde principalmente à implementação dos mecanismos de comunicação e às funções primitivas para o acesso ao sistema local de arquivos, corresponde, no caso do NPFS, a cerca de 12% do código total do sistema, ou seja, 88% do código do NPFS foi gerado automaticamente pelo compilador Ec.

6.1.4 Decisões de Implementação

Alguns aspectos da implementação são conduzidos de acordo com as características que se deseja incorporar ao sistema que está sendo implementado. No caso de um sistema de arquivos paralelos distribuídos, deve ser considerado o número de *sockets* associados a um processo, a localização e a falha de processos.

Número de sockets nos processos

A maneira mais simples de implementação do sistema de comunicação é criar apenas um *socket* em cada processo. Entretanto, dependendo do sistema a ser implementado, um número maior de *sockets* pode ser mais conveniente.

No caso específico de um sistema de arquivos centralizado, o processo Cliente precisa se comunicar com um processo Mestre e diversos outros processos Servidor. O uso de apenas um

socket para o acesso aos dados pode tornar a comunicação um gargalo, pois vários Servidores enviariam dados simultaneamente ao mesmo *socket* do Cliente. Para otimizar o desempenho na escrita e leitura de arquivos, *sockets* extras foram criados, cada um associado a um dos Servidores envolvidos.

Índices dos Processos

Na especificação Estelle do sistema cada processo Cliente e Servidor é referenciado por um índice (um número inteiro maior que zero), que é utilizado como identificador de processos durante a requisição de serviços. Esse índice possibilita, por exemplo, aos Servidores identificar no sistema os Clientes que lhes solicitam serviços. Por ser único na arquitetura, o único processo não indexado é o Mestre.

Ao se implementar o Sistema a referência por índices deve ser substituída por endereços de rede. Entretanto os índices não podem ser totalmente descartados, já que são usados internamente pelos processos do sistema. A solução é realizar um mapeamento entre índices e endereços para o acesso remoto de processos.

O *Ug* é capaz de gerar uma implementação automática do sistema de comunicação para plataformas *Unix*. Comentários especiais inseridos na parte de iniciação dos módulos, como demonstrado na Figura 45 para o módulo Mestre, indicam o nome do *host* e a porta a ser usada na comunicação. Nessa abordagem o número de processos Cliente e Servidor que participarão do sistema, deve ser definido estaticamente na especificação. Como após essa definição não é possível a inclusão de novos processos e nem a exclusão dos vigentes, os índices estarão sempre associados aos mesmos processos.

```
initialize      { criacao estatica das ocorrencias }
   begin
      ...
      { iniciação do Mestre }
      ($@$, "host_name", num_porta }
      init Modulo_Mestre with Corpo_Modulo_Mestre;
      ...
    end;
```

Figura 45: Definição da estação e da porta para o processo Mestre

Para ativar os processos distribuídos, o *Ug* cria dois processos especiais: o *exec_server*, que será executado junto a cada processo do sistema, e o *supervisor*, que é responsável pela

ativação dos processos distribuídos através do envio de mensagens aos seus *exec_servers* correspondentes.

Embora o *Ug* disponha de uma estratégia genérica para a implementação semi-automática de especificações *Estelle*, a implementação resultante fica muito dependente das definições estáticas de endereços e do número de processos envolvidos. No caso do sistema global o número de Servidores permanece fixo durante todo o tempo de execução do sistema. Entretanto a concepção original desse sistema permite a ativação e a desativação dinâmica de Servidores.

A fim de contornar essas limitações, foram criados meios de associar, em tempo de execução, os índices dos processos aos seus endereços de rede. Na prática isso permite que novos Usuários e Servidores se integrem ao sistema já em funcionamento. Essa abordagem também dispensa a utilização de processos extras tais como o *exec_server* e o *supervisor*.

Para que um processo Servidor passe a integrar o sistema, este precisa ser ativado pelo Mestre. Após a ativação cada Servidor passa a ter um índice associado ao seu endereço de rede. Essas informações, armazenadas pelo Mestre, ficam à disposição dos Clientes, que as utilizam para acessar diretamente os Servidores numa operação de escrita ou leitura de dados.

Um índice pertence a um Servidor até o instante em que este é desativado do sistema. Nesse caso o índice liberado pode ser entregue a outro Servidor, que vier a ser ativado no futuro. A liberação de índices é importante pois o número de Servidores, que podem fazer parte simultaneamente do sistema, é limitado e definido pelo Mestre.

Procedimento semelhante é aplicado em relação aos índices dos Clientes, com a diferença de que estes não precisam se cadastrar previamente para usar o sistema. A definição de seus índices é realizada no primeiro contato com o Mestre numa operação de abertura, remoção ou renomeação de arquivo. Como os Clientes também não são desvinculados explicitamente, a associação do índice permanece enquanto as operações de remoção ou renomeação não terminam, ou enquanto o Cliente tiver algum vínculo com o Mestre ao manter arquivos abertos no sistema. Assim como ocorre com os Servidores, o número de Clientes associados simultaneamente ao Mestre é limitado.

Falha de processos

Eventualmente processos distribuídos podem falhar e ficar desativados por algum tempo ou indefinidamente, criando situações que não podiam ser simuladas na especificação global. As conseqüências da indisponibilidade de processos devido a falhas são discutidas na seção

subseqüente, que aborda as simulações realizadas sobre a especificação com as funções primitivas e de rede implementadas.

6.2 Simulação com as Funções de Rede e das Funções Primitivas Implementadas

Após a divisão da especificação inicial e a implementação dos mecanismos de comunicação, há a necessidade de averiguar o correto funcionamento dos *sockets* e das funções primitivas, que não foram avaliados nas simulações realizadas até este estágio.

O simulador *Edb* é capaz de incorporar, ao código gerado para simulação do sistema, o código C que implementa as funções primitivas e os mecanismos de comunicação, permitindo a execução controlada dos processos distribuídos. Isso possibilita, por exemplo, a execução de uma função primitiva a partir do *Edb*. Do mesmo modo os mecanismos de comunicação podem ser acionados, permitindo aos componentes do sistema trocar mensagens usando os protocolos de rede durante a sessão de simulação.

As implementações dos processos obtidas são completas e podem ser executas sem o auxílio de um simulador. Entretanto o uso do *Edb* permite simular falhas de rede, que são impossíveis de ocorrer nesse estágio do projeto, onde todos os processos estão sendo executados num mesmo computador. Soma-se a este o fato da execução ser conduzida passo-a-passo, permitindo, por exemplo, verificar o funcionamento dos *sockets* tão logo mensagens sejam enviadas ou recebidas.

Mensagens corrompidas ou não entregues corretamente geram resultados errados, cujas causas seriam difíceis de detectar num teste do tipo caixa-preta, onde somente o resultado final do teste é verificado. Com o auxílio do simulador, é possível inspecionar as mensagens que são enviadas e recebidas pelos *sockets* antes que estas sejam entregues aos módulos destinatários. Após a utilização do simulador para a certificação do correto funcionamento das funções implementadas, o sistema poderá ser executado num ambiente operacional, onde serão aplicados testes baseados nos cenários de simulação já desenvolvidos.

Para a simulação distribuída do NPFS um processo Cliente, um Mestre e ao menos um Servidor tiveram sua execução simulada em sessões distintas do Edb, porém todas conduzidas no mesmo computador. A cada processo foram associadas portas de *sockets* diferentes.

Macros foram criadas para a simulação distribuída das operações de abertura, fechamento, remoção e renomeação de arquivos, além da escrita e leitura de dados nos Servidores. O principal

foco das novas simulações foi a comunicação por *sockets*, uma vez que o funcionamento dos processos já havia sido testado anteriormente.

Por exemplo, para a avaliação da operação de abertura de arquivo foram criadas macros para serem executadas no Cliente, no Mestre e nos Servidores. Para o Cliente bastou a definição de duas macros: a primeira para comunicação sem falhas, onde o Cliente obtém a resposta do Mestre independentemente se a operação foi bem ou mal sucedida; a segunda para falhas na comunicação (do Cliente para o Mestre ou vice-versa), obrigando o Cliente a retransmitir a requisição até que houvesse o contato do Mestre ou até que o número máximo de retransmissões fosse atingido.

Para o Mestre foi criado um segundo conjunto de cenários compreendendo quatro macros: a primeira garante o contato tanto com os Servidores envolvidos com a operação, quanto com o Cliente que a solicitou; a segunda simula a perda de mensagens entre Mestre e Servidores, onde o Mestre deve retransmitir o pedido de abertura até conseguir contactar os Servidores ou até desistir do contato (falha da operação); a terceira simula a perda de comunicação entre o Mestre e o Cliente, obrigando o Cliente a retransmitir a requisição de serviço; a quarta simula a perda de mensagens tanto entre o Mestre e os Servidores quanto entre o Mestre e o Cliente, reunindo as funcionalidades das últimas duas macros descritas.

Do lado do Servidor foram definidas duas macros: a primeira retorna a resposta à solicitação do Mestre em relação à abertura de arquivo (com ou sem êxito); a segunda simula a perda de mensagens entre o Servidor e o Mestre, o que leva o Mestre a retransmitir o pedido. Todas as situações referentes às macros discutidas estão descritas na Figura 46.

Cliente	Mestre	Servidor
Sit_1: Cliente contacta Mestre sem problemas de Rede; Sit_2: perda de mensagens entre Cliente e Mestre	Sit_1: Mestre contacta Servidores sem problemas de Rede; Sit_2: perda de mensagens entre Mestre e Servidores; Sit_3: Mestre contacta Servidores, mas resposta ao cliente se perde; Sit_4: perda de mensagens entre Mestre e Servidores e entre Mestre e Cliente	Sit_1: Servidor responde ao Mestre (OK/ERRO); Sit_2: perda de mensagens entre Servidor e Mestre.

Figura 46: Situações para simulação distribuída

Os cenários de simulação foram formados através da combinação das situações, definidas pelas macros distribuídas, entre os processos participantes de uma operação. No caso de abertura de arquivo foram definidos 18 cenários, dentre estes os casos onde há falha irrecuperável da rede, quando o Mestre contacta os Servidores e quando o Cliente contacta o Mestre. Operações de fechamento, renomeação e remoção de arquivos definem um número maior de cenários, pois podem ser realizadas síncrona ou assincronamente. No que se refere à comunicação, as operações de leitura e escrita de dados geraram um número menor de cenários, pois somente dois tipos de processos (Cliente e Servidor) estão envolvidos nessas operações.

6.3 Considerações sobre a Simulação com as Funções de Rede e Funções Primitivas Implementadas

O principal objetivo das simulações da especificação distribuída foi averiguar se as funções, que iniciam e utilizam os *sockets* para o acesso à rede, estavam implementadas corretamente. Entretanto novas situações, que não ocorrem numa especificação global, puderam ser analisadas.

Na especificação inicial, antes da divisão em especificações isoladas para cada processo do sistema, o número de Clientes e Servidores era fixo durante toda a simulação e esses processos sempre estavam disponíveis. Os Servidores eram pré-determinados e registrados no Mestre na iniciação do sistema e os Clientes sempre tinham à disposição ao menos um Servidor.

Nos testes distribuídos verificou-se a possibilidade de o Cliente tentar criar um arquivo, sem que nenhum Servidor ainda estivesse cadastrado no sistema, o que levaria a um erro de execução no processo Mestre. Da mesma forma um Servidor pode a qualquer instante ser descadastrado e se tornar indisponível, o que causaria outra condição de erro caso um Cliente tentasse utilizá-lo.

Novas funções foram inseridas no Mestre, para alertar o Cliente da inexistência de Servidores para realização de serviços. Passou-se também a verificar se a lista de Servidores, que é usada na criação de um novo arquivo distribuído e entregue ao Mestre pelo Cliente, possui todos os Servidores registrados. Se um ou mais Servidores da lista não estiver disponível, o Mestre cria o arquivo sobre aqueles que estão cadastrados e retorna ao Cliente uma lista válida.

Novos aspectos relativos à comunicação também foram examinados. Na especificação global a ausência de resposta a uma requisição era sempre atribuída a uma falha da rede e nunca à falha de um processo. Numa abordagem distribuída processos podem ter o seu funcionamento interrompido, fazendo com que seus estados e informações internas sejam perdidos.

Nos testes foi comprovada a importância de se preservar os índices de identificação de Servidores e Clientes, que eventualmente falham e podem retornar posteriormente. Processos, cujos índices são alterados indiscriminadamente, ficam inacessíveis aos demais processos que tentam contatá-los pelo índice antigo.

É de responsabilidade do processo Mestre, que registra e mantém estruturas de dados que associam índices e endereços, recadastrar Servidores e Clientes de modo a preservar corretamente seus índices. Por essa razão e por manter as informações relativas aos arquivos distribuídos, a falha do Mestre é considerada a mais crítica.

Nas simulações realizadas nos Capítulos 4 e 5, a perda de mensagens ocorria sempre entre o processo requerente de serviço e o processo respondedor. Como o objetivo prioritário era testar as transições que retransmitiam requisições, não havia diferença se a mensagem perdida era a requisição ou a resposta, pois nos dois casos as mesmas transições de retransmissão eram acionadas.

Porém nos testes dos *sockets* é importante determinar se a requisição e a resposta estão sendo corretamente encaminhadas. Por essa razão foram consideradas situações diferentes para os casos onde mensagens são perdidas entre o requerente e o respondedor e entre o respondedor e o requerente, ampliando assim a cobertura dos testes da Rede.

6.4 Implantação e Desempenho do Sistema

Para o teste e a avaliação de desempenho do sistema foi usado um conjunto de computadores PC, interligados por uma rede *Fast Ethernet* (100 MB/s). Cada máquina dispunha de processadores Celeron 433 MHz, 128 *MBytes* de memória RAM e discos de 80 GB. Os computadores utilizavam software Red Had Linux 8.0.

Os primeiros testes foram voltados à verificação da correta implementação dos mecanismos de comunicação e das funções primitivas. Operações para abertura, fechamento, renomeação e remoção de arquivos distribuídos, em um número variável de Servidores, foram realizadas, tendo como base os cenários de simulação criados para o sistema centralizado.

Um programa para computar o tempo gasto, entre o início e o término de uma operação de leitura ou escrita de dados, foi empregado para a avaliação do desempenho de operações seqüenciais de E/S sobre um arquivo paralelo. Os testes foram executados diversas vezes para atestar a consistência dos dados. Foi considerado, como medida de desempenho, o tempo médio de resposta das diversas realizações de um mesmo teste.

6.4.1 Entrada e Saída de dados

Para a avaliação de desempenho de leitura e escrita de dados foi usado um arquivo de 512 MB, suficientemente grande para exceder a capacidade de *cache* de disco do sistema operacional, forçando assim o acesso aos discos. Para cada teste a unidade de distribuição foi obtida dividindo-se o tamanho da requisição pelo número de Servidores, a fim de que todos participassem das operações.

O tamanho das requisições iniciava em 512 *Bytes* e variava entre 1KB e 64 KB. Requisições maiores não foram empregadas porque seriam fragmentadas pelo protocolo de comunicação utilizado (UDP). Os testes foram realizados sobre grupos de 1, 2, 4, 6 e 8 Servidores.

No módulo Usuário foram inseridas funções, que solicitam a leitura ou escrita de dados e que computam o tempo transcorrido durante a execução dessas operações. O Usuário foi executado num computador que também funcionava como Servidor de dados, possibilitando assim a avaliação de desempenho de um Servidor local sem necessidade de acesso à rede.

O número total de *bytes*, envolvidos numa operação, dividido pelo tempo transcorrido para a sua realização, resultou na taxa de transferência para cada teste. Os resultados obtidos, demonstrados nas Figuras 47 e 48 para a escrita e leitura de dados respectivamente, permitem comparar o sistema implementado semi-automaticamente com um equivalente implementado manualmente (GUARDIA, 1999), cujas análises de desempenho são apresentadas nas Figuras 49 e 50.

Para pequenas requisições de dados o sistema apresentou desempenho inferior àquele obtido sem a utilização de métodos formais, que possui um código mais otimizado. Porém com o aumento do tamanho da requisição, o desempenho se equipara ao sistema original e tende a se estabilizar em requisições de 44KB a 64KB. Requisições maiores precisam de um número menor de acesso aos discos para transferir os dados, o que contribui para o aumento da taxa de transferência.

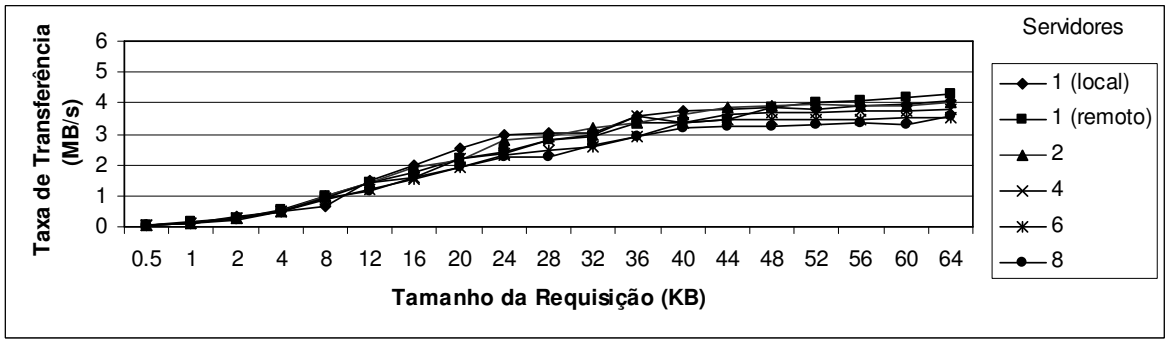


Figura 47: Escrita de dados para o sistema implementado semi-automaticamente

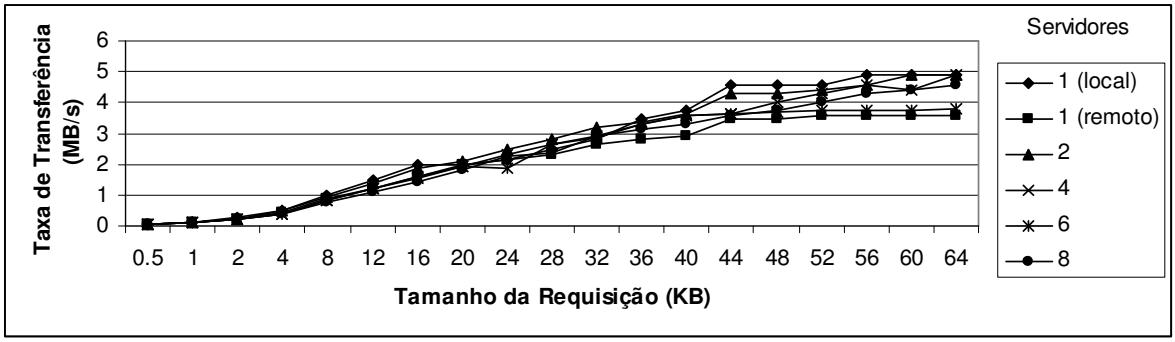


Figura 48: Leitura de dados para o sistema implementado semi-automaticamente

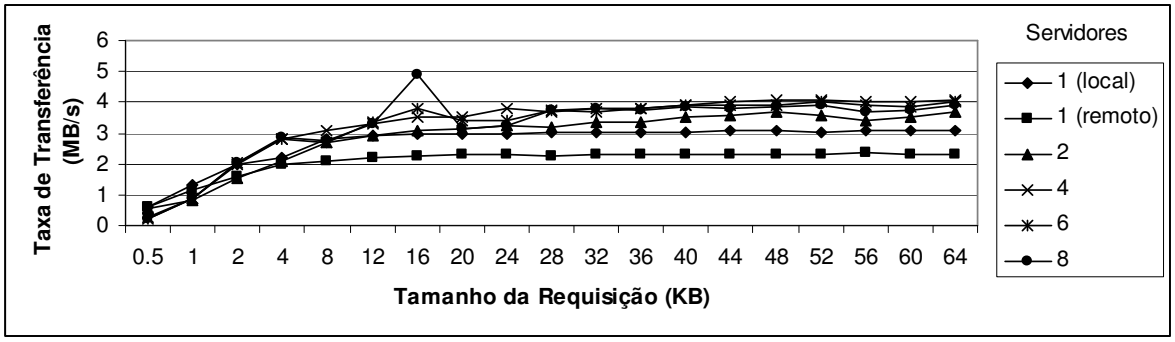


Figura 49: Escrita de dados para o sistema implementado manualmente

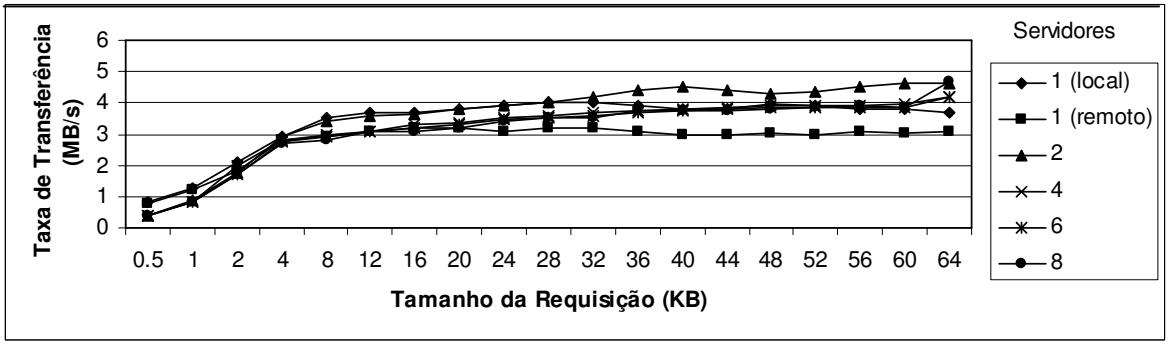


Figura 50: Leitura de dados para o sistema implementado manualmente

Para fazer um uso mais eficiente do *throughput* oferecido pela rede, requisições de tamanho inferior a 44KB poderiam ser reunidas numa única solicitação de leitura ou numa única transferência de dados para escrita. Essa otimização poderia ser realizada pela própria aplicação, que acessa os dados nos discos, ou ser oferecida pelo sistema de arquivos.

Por outro lado, as aplicações que utilizam sistemas de arquivos paralelos distribuídos tendem a manipular um grande volume de dados, característica que se reflete no tamanho maior das requisições de E/S. Isso minimiza o impacto que o baixo desempenho para pequenas requisições apresentado pelo sistema implementado semi-automaticamente poderia ter sobre o desempenho final dessas aplicações.

O emprego de um conjunto maior de servidores para o armazenamento de dados aumenta o paralelismo das operações de escrita e leitura. Porém a competição pela rede de comunicação compartilhada para atender as requisições dessas operações impossibilita um aumento significativo no desempenho ao se disponibilizar mais servidores aos Clientes. Essa característica é observada nas duas versões do sistema por ser independente da otimização da implementação.

6.4.2 Análise da Implementação

A escolha do protocolo de comunicação é fundamental para o desempenho do sistema. Neste exemplo foi empregado o protocolo UDP, que permite a comunicação sem o estabelecimento de conexão e a operação assíncrona entre os componentes do sistema. Esse fato demonstra que as decisões tomadas pelo implementador, com relação à infraestrutura de comunicação, têm grande influência no desempenho final de um sistema concebido a partir do *framework*. As demais escolhas apresentaram uma influência menor mas não negligenciável. Funções que particionam os dados entre Servidores, por exemplo reúnem pequenos blocos contíguos destinados a um mesmo Servidor de maneira que possam ser encaminhadas numa única transmissão, também colaboram com o desempenho, uma vez que diminuem o número de mensagens trocadas entre Clientes e Servidores.

Os dois sistemas especificados, tanto o descentralizado quanto sua versão centralizada, acessam os dados nos discos da mesma forma, ou seja, os dados perfazem o mesmo caminho desde a solicitação do Usuário até a resposta recebida. Assim sendo a implementação, no que diz respeito às operações de E/S, teria o mesmo desempenho se o sistema descentralizado tivesse sido escolhido como estudo de caso.

A complexidade dos sistemas de arquivos reside nas demais operações (abertura, fechamento, renomeação e remoção), que necessitam consultar um processo coordenador centralizado ou entrar em contato com um Meta-Servidor. Nesses casos há um maior número de transições envolvidas e um maior número de troca de mensagens de controle entre processos via rede, comparado ao que ocorre com as operações de leitura e escrita.

7 Conclusões

A abordagem apresentada neste trabalho, para o desenvolvimento de sistemas de arquivos paralelos distribuídos, é constituída de cinco fases principais: projeto, especificação formal, validação da especificação, implementação semi-automática e teste da implementação.

A fim de generalizar a abordagem foi apresentado um *framework*, que descreve a estruturação básica desses sistemas e que pode ser complementado com características próprias, as quais distinguem os diversos sistemas de arquivos existentes. O *framework* foi descrito com os conceitos e regras estruturais de uma TDF, servindo de modelo para a concepção de especificações para outros sistemas mais elaborados.

A escolha da TDF mais adequada para a construção do *framework* envolveu o estudo de um meta-modelo, englobando as principais características estruturais e comportamentais, que devem estar presentes num modelo de projeto para o desenvolvimento de sistemas de arquivos paralelos distribuídos. As representações dessas características, encontradas nas quatro TDFs estudadas (LOTOS, E-LOTOS, SDL e Estelle), demonstraram que Estelle é a mais apropriada para um modelo de projeto de frameworks para esses sistemas. Além de apresentar conceitos que facilitam a especificação desse tipo de sistema, Estelle usa a linguagem de programação Pascal, facilitando assim a compreensão das especificações por parte dos projetistas. Finalmente, por ser uma TDF orientada para implementação, é mais fácil gerar semi-automaticamente implementações a partir de especificações Estelle do que a partir de especificações realizadas nas outras TDFs estudadas.

Partindo desse *framework* dois sistemas, representativos das duas grandes categorias de sistemas de arquivos distribuídos, foram especificados. O primeiro, denominado *Galley*, possui a mesma estrutura descentralizada do *framework* e sua especificação envolveu o preenchimento das lacunas do *framework* com as MEFEs dos processos que denotam seu comportamento. O segundo sistema, denominado NPFS, exigiu a alteração da estrutura do *framework*, tendo sido incluído um novo processo centralizador de serviços. Comparações, entre as diferenças e similaridades observadas durante o processo de especificação, foram realizadas para avaliar a flexibilidade do *framework*.

Para validar a especificação formal dos dois sistemas foi criado um conjunto de situações que refletem o comportamento dos processos durante a realização dos serviços. Todas as possíveis

combinações de situações levaram à criação de um conjunto de cenários, onde cada cenário representa a realização de uma operação completa, isto é, do início ao seu término. Embora não seja possível simular exaustivamente um sistema, as simulações realizadas demonstraram que os sistemas gozam de propriedades importantes, tais como recuperação de impasses, ausência de laços improdutivos, ausência de interações não tratadas e término apropriado, o que outorga um alto grau de confiabilidade ao projeto desse sistema.

Para demonstrar a obtenção da implementação de um sistema, a partir da sua especificação formal, foi escolhido o NPFS por este dispor de um maior número de processos. Após a etapa de simulação da especificação global do NPFS, que contemplava o sistema como um todo, três especificações correspondentes aos processos Cliente, Mestre e Servidor foram criadas com o auxílio de ferramentas. As funções *primitive* de Estelle e os mecanismos de comunicação foram codificados e ligados ao código C correspondente, gerado pelo compilador *Ec*, dando origem às implementações dos processos. Estes puderam ser distribuídos sobre um conjunto de computadores, conectados através de uma rede local.

Um novo conjunto de cenários de simulações foi concebido, focado principalmente na comunicação e interação dos componentes distribuídos do NPFS. O simulador foi novamente usado, pois as simulações anteriores foram realizadas em apenas um computador, o que impossibilitava a simulação de problemas decorrentes da rede (e.g., atraso e perda de mensagens).

Como o código de implementação pode ser gerado semi-automaticamente a partir da especificação, o desenvolvimento e teste dos sistemas implementados por esse método levou muito menos tempo que o de uma implementação codificada manualmente. Testes de desempenho mostraram que os sistemas desenvolvidos das duas formas, manual e semi-automática, apresentam o mesmo desempenho para a solicitação de grandes requisições de dados. A desvantagem para a adoção dessa abordagem de projeto de sistemas de arquivos paralelos distribuídos seria o baixo desempenho apresentado nas situações em que o acesso aos dados é feito com pequenas requisições. Porém as aplicações que utilizam esse tipo de sistema realizam grandes requisições para acesso aos dados armazenados nos discos. Isso demonstra que os ganhos em relação aos tempos de desenvolvimento e de teste, obtidos com o emprego da abordagem proposta neste trabalho, são mais importantes do que a otimização do código de implementação obtida através de métodos tradicionais de desenvolvimento.

7.1 Contribuições

A principal contribuição deste trabalho está na inclusão de técnicas de descrição formal ao ciclo de desenvolvimento de sistemas de arquivos paralelos distribuídos. Dentre as vantagens dessa abordagem destacam-se:

- obtenção de uma descrição do sistema concisa e livre de ambigüidades, que não é possível de se obter com métodos tradicionais que usam técnicas semi-formais e linguagem natural para descrever sistemas;

- detecção e correção de erros do sistema, via simulação da especificação, antes da fase de implementação. A presença de erros que podem levar, por exemplo, a *deadlock*, são acusados e corrigidos durante a simulação, diminuindo assim as chances de aparecimento de erros na implementação;

- Implementação semi-automática com o auxílio de ferramentas, que convertem a especificação em código de uma linguagem de programação.

Para generalizar a abordagem foram estudadas as características comuns, compartilhadas por diversos sistemas de arquivos distribuídos existentes, o que levou à criação de a um *framework* conceitual para esses sistemas. A adoção do *framework*:

- Disponibiliza, ao projetista de sistemas de arquivos, um ponto de partida para a especificação e implementação desses sistemas;

- permite criar uma especificação de uma maneira rápida e eficiente, sendo que o maior trabalho é o preenchimento das lacunas deixadas pelo *framework*, que correspondem principalmente às MEFEs dos processos. O preenchimento sistemático do *framework* diminui a ocorrência de erros de especificação.

Os dois sistemas especificados também podem ser usados como modelo. A arquitetura do NPFS, divergente da arquitetura básica do *framework* conceitual, poderia ser considerada um modelo para sistemas de arquivos centralizados.

7.2 Projetos Futuros

Dentre os projetos que poderiam ser desenvolvidos, estão a criação de novos *frameworks* para sistemas com arquiteturas mais complexas que as apresentadas neste trabalho, incluindo, além do centralizador de serviços, novos processos.

Outras TDFs poderiam ser usadas na criação dos *frameworks*. Essas alternativas poderiam ser dispostas em uma matriz de opções de *frameworks*, que ficaria à disposição de especificadores e implementadores.

Neste trabalho a implementação do acesso à rede de comunicação usou apenas uma das muitas alternativas possíveis. Outros mecanismos para a troca de mensagens entre processos, tais como *Message Passing Interface (MPI)* (MPI, 2005), poderiam ser empregados. A escolha reside em questões como desempenho, portabilidade da aplicação e simplicidade de implementação.

Operações mais sofisticadas, para o acesso paralelo aos arquivos, podem vir a ser incluídas em versões futuras do *framework*, na medida em que se tornarem padrões entre os sistemas de arquivos. A documentação da maneira como a adição de novas primitivas de serviço afeta o comportamento dos processos do sistema poderia ser futuramente disponibilizada para servir como guia para que projetistas, especificadores e implementadores adaptassem os *frameworks* disponíveis para refletir essa inclusão.

LISTA DE REFERÊNCIAS

AGERWALA, T.; MARTIN, J. L.; MIRZA, I. H.; SADLER, D. C.; DIAS, D. M.; SNIR, M. **SP2 System Architecture, Scalable Parallel Computing**. Volume 34, N^{o} 2, 1995.

AMER, P. D.; SETHI, A.S.; FECKO, M. **Formal Design and Testing of Army Communication Protocols Based on Estelle**, Proc. 1st ARL/ATIRP Conf, College Park, January 1997, p. 107-114.

AMER, P. D.; SETHI, A.S.; FECKO, M.; UYAR, M.Ú.; DZIK, T.; MENELLl, R.; MCMAHON, M. **Using Estelle to Evolve MIL-STD 188-220**, Proceedings of 1st. International Workshop on FDT Estelle, 1998.

BAGRODIA, R.; CHIEN, A.; HSU, Y.; REED, D. **Input/Output: Instrumentation, Characterization, Modeling and Management Policy**. Scalable I/O Initiative Working Paper No 2. Performance Evalution Working Group of the Scalable I/O Initiative, 1994.

BARBOSA, C. B. **Frameworks for Implementing Protocols: a Model Based Approach**, Ph. D. Thesis, Enschede, The Netherland, 2001

BASIC REFERENCE MODEL, International Organization for Standardization, ISO/IEC 7498, 1984.

BERSHAD, B. **Scalable I/O Initiative**. Working Paper No. 4, 1994.

BERSHAD, B.; BLACK D.; DEWITT, D.; GIBSON, G.; Li, K.; PETERSON, L.; SNIR, M. **Operation system support for high-performance parallel I/O systems**. Relatório Técnico CCSF-40, Scalable I/O Initiative, Caltech Concurrent Supercomputing Facilities, Caltech, 1994.

BOLOGNESI, T.; VAN DE LAGEMAAT, J.; VISSERS C. **LOTOSphere: Software Development with LOTOS**, Kluwer Academic Publishers, 1995.

BORCOCI, E. **Performance evaluations for Estelle Specifications**, Research Raport Nr. RR 98 10 02, INT, 1998.

BUDKOWSKI, S.; DEMBINSKI, P. **An Introduction to Estelle: A Specification Language for Distributed Systems**, Computer Network and ISDN Systems Journal, vol. 14, Nb.1, 1988.

BULL S.A., Louveciennes. **Ec – User Reference Manual** (version 4.3), França, 2000a.

BULL S.A., Louveciennes. **Edb – User Reference Manual** (version 4.3), França, 2000b.

BULL S.A., Louveciennes. **Ug – User Reference Manual** (version 4.3), França, 2000c.

CATRINA, I; CATRINA, O. **Studying the Connection-Oriented Communications by Simulation with Estelle Specifications** – Research Report Nr. RR 98 10 01, Politehnica University Bucharest – 1998.

CHOUDHARY, A.; FOSTER, I.; FOX, G.; KENNEDY, K.; KESSELMAN, C.; KOELBEL, C; SALTZ, J.; SNIR, M. **Languages, Compilers and runtime Systems Support for Parallel Input-Output**. **Scalable I/O Initiative**. Working Paper No 3, Compilers and Languages Working group of the Scalable I/O Initiative, 1994.

CORBETT, P. F.; FEITELSON, D. G. **Design and Implementation of the Vesta Parallel File System**. Proceedings of the Scalable High-Performance Computing Conference, pp 63-70, 1994.

CORBETT P.F.; FEITELSON, D. G.; PROST, J.; ALMASI G. S.; BaYLOR S. J.; BOLMARCICH A. S.; HSU, Y.; SATRAN, J.; SNIR M.; CHI, M.; COLAO, R.; HERR, B.; KAVAKY, J.; MORGAN, T. R.; ZLOTEK, A. **Parallel File Systems dor the IBM SP Computers**. IBM System Journal, Vol. 34, N^{o} 2, pp 222-248, June 1995.

CORBETT, P. F.; PROST, J.; DEMETRIOU, C. G.; GIBSON, G.; RIEDEL, E.; ZELINKA, J.; CHEN, Y.; FELTEN, E.; LI, K.; HARTMAN, J.; PETERSON, L.; BERSHAD, B.; WOLMAN, A.; AYDT, R., **Proposal for a Common Parallel File System Programming Interface**. 1996.

CORMEN, T.H.; KOTZ, D. **Integrating Theory and Practice in Parallel File Systems**. Anais PCS-TR93-188 do 1993 DAGS/PC Symposium, Hanover,NH, 1993.

DEL ROSARIO, J.M.; CHOUDHARY, A.N. **High-Performance I/O for Massively Parallel Computers: Problems and Prospects**. IEEE Computer, março de 1994.

DEPLANCHE, A.; OLLIVE, F.; TRINQUET, Y. **Using Estelle for the Validation of a Fault-Tolerance Management Protocol**. Proceedings of 1st. International Workshop on FDT Estelle, 1998.

DIAZ, M.; ANSART, J. P.; COURTIAT, J. P.; AZEMA, P.; CHARI, V. **The Formal Description Technique Estelle**, North-Holland, 1989.

DIEFENBRUCH, M.; HINTELMANN, J.; MÜLLER-CLOSTERMANN, B., **The QUEST-Approach for the Performance Evaluation of SDL-Systems**. Proc. IFIP TC6/6.1 Int. Conf. on Formal Description Techniques IX, Kaiserslautern, Germany, pp. 229 – 244, 1996.

DOHERTY, S.; GROVES, L.; LUCHANGCO, V.; MOIR, M., **Formal Verification of a Practical Lock-Free Queue Algorithm**. Formal Techniques for Networked and Distributed Systems – FORTE, 24th IFIP WG 6.1 International Conference, Madrid, Spain, September, 2004.

EDT - ESTELLE DEVELOPMENT TOOLSET, version 4.3, Institut National des Télécommunications Software (INT) - Networks Department, France, 2000.

EHRIG, H.; MAHR, B. **Fundamentals of Algebraic Specification**. Bull. Euro. Assoc. Theoret. Comp. Sci., 6, 1985.

ESPRIT PROJECT 1265 – SEDOS, Bruxelas. **EWS User´s manual**. Bélgica, 1989.

FAERGEMAND, O. , OLSEN, A. **Introduction to SDL-92**, DK- 2970 Horsholm, Denmark, 1992.

FECKO, M. A.; UYAR, M. U.; AMER, P. D.; SETHI, A. S.; DZIK, T.; MENELL, R. MCMAHEN, M., **A Success Story of Formal Description Techniques: Estelle Specification and Test Generation for MIL-STD 188-220**. Computer Communications, Vol. 23, n. 12, pp. 1196 – 1213, June 2000.

FOWLER, M.; SCOTT, K. **UML Distilled – Applying the Standard Object Modeling Language**. Addison-Wesley, 1997.

GARAVEL, H.; LANG, F.; MATEESCU, R. **An Overview of CADP 2001**, Raport Technique n^{o} 0254, Institut National de Recherche en Informatique et em Automatique (INRIA), Décembre 2001.

GARCIA, F.; CALDERÓN, A.; CARRETERO, J.; PÉREZ, J. M.; FERNÁNDEZ, J. **A New Approach to the Construction of Parallel File Systems for Clusters**. International Conference on Advances in Infrastructure for Electronic Business, Education Science, Medicine, and Mobile Technologies on the Internet (SSGRR 2002s). L' Aquila, Italia. 29 de julio 4 de agosto 2002.

GIBSON, G. A. ; STODOLSKY, D.; CHANG, F. W.; COURTRIGHT, W. V.; DEMETRIOU, C. G.; GINTING, E.; HOLLAND, M.; MA, Q.; NEAL, L.; PATTERSON, R. H.; SU, J.; YOUSSEF, R.; ZELENKA, J. **The Scotch parallel storage system**. In COMPCON '95. Technologies for the Information Superhighway, pp 403-410, Los Alamitos, CA, 1995.

GUARDIA, H. C. **Considerações Sobre as Estratégias de um Sistema de Arquivos Paralelos Integrado ao Processamento Distribuído**. Tese de Doutorado, EPUSP 1999.

GUARDIA, H. C.; SATO, L. M. **NPFS: Um Sistema de Arquivos Paralelos em Rede.** In XVII Simpósio Brasileiro de Redes de Computadores, 1999, Salvador

HARTMAN, J. H.; OUSTERHOULT, J. K. **Zebra: A Striped Network File System**. In Proceedings of the Fourteenth ACM Symposion on Operation Systems Principles, pp. 29-43, 1993.

HAYDAR, M.; PETRENKO, A.; SAHRAOUI, H. **Formal Verification of Web Applications Modeled by Communicating Automata**. Formal Techniques for Networked and Distributed Systems – FORTE, 24th IFIP WG 6.1 International Conference, Madrid, Spain, September, 2004.

HOARE, C. A. R. **Communicating Sequential Processes**. Prentice-Hall, 1985.

HUBER Jr., J. V.; ELFORD, C. S.; REED, D. A.; CHIEN A. A.; BLUMENTHAL, D. S. **PPFS: A High Performance Portable Parallel File System**. In Proceedings of the 9th ACM International Conference on Supercomputing (July 1995), pp. 385-394.

INRIA Rhône-Alpes, VASY team. **TRAIAN: A Compiler for E-LOTOS/LOTOS**. May 2003, France.

ISO. **E-LOTOS: Enhancements to LOTOS**. ISO/IEC 15437, 2001.

ISO. **Estelle: A Formal Description Technique Basead on an Extended State Transition Model**. ISO IS 9074, 1989.

ISO IS 7185. **Programming Language Pascal**, 1983.

ISO. **LOTOS: A Formal Description Technique Basead on the Temporal Ordering of Observational Behaviour**. ISO IS 8807, 1989.

ITU-T. **Specification and Description Language (SDL)**. Recommendation Z.100, International Telecommunications Union - Telecommunication, 1996.

KAYNAR, D. K.; CHETTER, A.; DEAN, L.; GARLAND, S. J.; LYNCH, N. A.; WIN, T. N.; RAMIREZ-ROBREDO, A. **Simulating nondeterministic systems at multiple levels of abstraction**. Proceedings of Tools Day, held in conjunction with CONCUR 2002, Brno, Czech Republic, August 4, 2002.

LOPES DE SOUZA, W. **Estelle: uma técnica para descrição formal de serviços e protocolos de comunicação**. Revista Brasileira de Computação, Rio de Janeiro, vol.5 (1), pp. 33-44, jul/set, 1989.

MANTOVAN, U. **Especificação Formal e Validação de um Sistema de Arquivos Paralelos Distribuídos**. Dissertação de Mestrado, PPG-CC/UFSCar, 2000.

MESSAGE PASSING INTERFACE. LAM/MPI Parallel Computing. Disponível em: < www.lam-mpi.org >. Acesso em: 10 dez. 2005

MILNER, R. **Communication and Concurrency**. Prentice-Hall, 1989.

MINILITE USER MANUAL, University of Twente, The Netherlands, 2004.

MOYER, S. A.; SUNDERAM, V. S. **PIOUS: a scalable parallel I/O system for distributed computing environments**. In Proceedings of the Scalable High-Performance Computing Conference, pages 71-78, 1994.

NIEUWEJARR, N.; KOTZ, D., **The Galley Parallel File System**. Department of Computer Science, Dartmounth College, Hanover, NH 03755-3510, 1996.

OBJECT GEODE. Disponível em:
<www.telelogic.com/products/additional/objectgeode/index.cfm> . Acessado em 07 de abr. 2004.

OBJECT MANAGEMENT GROUP. Na Internet: http://www.omg.org.

POOLE, J.T. **Preliminary Survey of I/O intensive applications**. Technical CCSF-38, Scalable I/O Iniative, Caltech Concurrent Supercomputing Facilities, Caltech, 1994.

ROMANO, P.; ROMERO, M.; CICIANI, B.; QUAGLIA, F. **Validation of the Sessionless Mode of the HTTPR Protocol. Formal Techniques for Networked and Distributed Systems**. FORTE, 23rd IFIP WG 6.1 International Conference, Berlin, Germany, September, 2003.

SAFIRA SDL. Disponível em: <www.solinet.com/safire.htm>. Acesso em 07 de abr. 2004.

SIJELMASSI, R.; STRAUSSER, B. **NIST Integrated Tool Set for Estelle**. Proceedings of IPIP Conference on Formal Descriptions Techiques, FORTE 90, Madrid, Spain.

SIGHIREANU, M. **LOTOS NT User's Manual (Version 2.3)**, INRIA Rhône-Alpes, VASY team, May 2003, France, 107 pages.

SIJELMASSI, R.; STRAUSSER B. **The Portable Estelle Translator: an overview and user guide**. Technical Report NCSL/SNA – 91/2 , January 1991a

SIJELMASSI, R.; STRAUSSER, B. **The Distributed Implementation Generator:an overview and user guide.** Technical Report NCSL/SNA – 91/3 , January 1991b

SINDERELLA. Disponível em: <www.cinderella.dk/index.htm>. Acesso em 08 de abr. 2004.

SITE: SDL Integrated Tool Environment, Berlin University. Department of Computer Science, Research team system analysis. Disponível em: <www.informatik.hu-berlin.de/SITE/site.html.em>. Acesso em 10 de abr. 2004.

SPARX SYSTEMS' UML 2.0 TUTORIAL. **Introductory tutorial on the UML notation and usage in software engineering.** Disponível em: <www.sparxsystems.com.au/UML_Tutorial.htm>. Acesso em 11 de jul. 2005.

SUN MICROSYSTEMS' UML TUTORIALS. Disponível em: <http://developers.sun.com/prodtech/javatools/jsenterprise/learning/tutorials/index.jsp>. Acesso em 11 de jul. 2005.

TANENBAUM, A. S. **Distributed Operating Systems**. Prentice Hall International Editions – 1995.

TELELOGIC TAU SDL SUÍTE. Disponível em: <www.telelogic.com/products/tau/sdl/index.cfm>. Acesso em 10 de abr. 2004.

THE XEludo USER MANUAL, Environment LOTOS de l'Université d'Ottawa, Canada. Manual disponível em <http://lotos.site.ottawa.ca/eludo> . Acesso em 10 de abr. de 2004.

TURNER, K. J. **Using Formal Description Techniques - An Introduction to Estelle, LOTOS and SDL**. John Wiley & Sons, 1993.

TURNER K. J. **The Formal Specification Language LOTOS - A Course for Users**. Department of Computing Science, University of Stirling, Scotland, April 1996.

UNIFIED MODELING LANGUAGE - OMG - Object Management Group. Na internet: <www.uml.org.>. Acesso em 11 de nov. de 2005.

VAN EIJK, P. H. J.; VISSERS, C. A. V.; DIAZ, M. **The Formal Description Technique LOTOS**. North-Holland,1989.

VERDEJOO, A. **E-LOTOS Tutorial with Examples**. Dept. de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, April 2000.

WASSYNG, A.; LAWFORD, M. **Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project**. FME 2003: International Symposium of Formal

Methods Europe Proceedings, Pisa, Italy, LNCS Vol. 2805, Springer-Verlag, September 2003, 133-153.

WIN, T.N. ; ERNST, M. D.; GARLAND, S. J.; KAYNAR, D. K.; LYNCH, N. **Using simulated execution in verifying distributed algorithms**. Software Tools for Technology Transfer (2003) 4, pages 1-10. Proceedings of Fourth International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2003), New York, January 2003.

Anexo A: Diagramas de Transição de Estados

São apresentados abaixo os Diagramas de Transição de Estados para os processos Cliente, Servidor e Meta-servidor do Sistema de Arquivos Descentralizado

A.1 Diagrama de Transição de Estados do Processo Cliente

Estado Atual	Evento	Ação	Próximo Estado
Espera	gfs_open()	• oper = REG_FILE • Envia pedido ao metaservidor: reg_file() • N_RETRANS = 0 • Inicia *time-out*	Abrindo
	gfs_unlink() (ESPERA = 0)	• oper = REG_UNLINKING • Envia pedido ao metaservidor: reg_file_status_name() • N_RETRANS = 0 • Inicia *time-out*	Removendo
	gfs_unlink() (ESPERA = 1)	• Responde ao Usuário: confirm_gfs_unlink(OK) • oper = REG_UNLINKING • Envia pedido ao metaservidor: reg_file_status_name() • N_RETRANS = 0 • Inicia *time-out*	Removendo
	gfs_rename() (ESPERA = 0)	• oper = REG_RENAMING • Envia pedido ao metaservidor: reg_file_status_name() • N_RETRANS = 0 • Inicia *time-out*	Renomeando
	gfs_rename() (ESPERA = 1)	• Responde ao Usuário: confirm_gfs_rename(OK) • oper = REG_RENAMING • Envia pedido ao metaservidor: reg_file_status_name() • N_RETRANS = 0 • Inicia *time-out*	Renomeando
Abrindo	*Timeout* (N_RETRANS < MAX_RETRANS) && (oper = REG_FILE)	• N_RETRANS++ • Retransmite pedido ao metaservidor: reg_file() • Inicia *time-out*	Abrindo
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = REG_FILE)	• Responde ao Usuário: confirm_gfs_open(-1)	Espera
	confirm_reg_file (fid >= 0, Servidores)	• Armazena Servidores envolvidos • N_SERVERS = Nro de servidores envolvidos • N_CONFIRM = 0 • N_RETRANS = 0 • oper = OPENING_FILE • Envia pedido aos servidores: gfs_sopen() • Inicia *time-out*	Abrindo
	confirm_reg_file (fid < 0)	• Responde ao Usuário: confirm_gfs_open(-1)	Espera

Abrindo	confirm_sopen(OK) (N_CONFIRM+1 = N_SERVERS)	• Cancela *time-out* • Registra status do arquivo no metaservidor: reg_file_status(ARQ_ABERTO) • oper := REG_STATUS • N_RETRANS := 0	Abrindo
	confirm_sopen(OK) (N_CONFIRM+1 < N_SERVERS)	• Atualiza confirmação do servidor • N_CONFIRM++	Abrindo
	confirm_sopen(ERRO) (N_SERVERS > 1)	• Responde ao Usuário: confirm_gfs_open(-1) • Envia pedido de fechamento de arquivo aos servidores: gfs_sclose() • N_CONFIRM = 0 • N_RETRANS = 0 • RESP = 0 • oper = CLOSING_FILE • Inicia *time-out*	Fechando
	confirm_sopen(ERRO) (N_SERVERS = 1)	• oper = REG_STATUS • Registra arquivo fechado no metaservidor: reg_file_status(ARQ_FECHADO) • N_RETRANS = 0 • Inicia *time-out*	Abrindo
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = OPENING_FILE)	• N_RETRANS++ • Retransmite pedido aos servidores que não responderam: gfs_sopen() • Inicia *time-out*	Abrindo
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = OPENING_FILE)	• Responde ao Usuário: confirm_gfs_open(-1) • Envia pedido de fechamento de arquivo aos servidores que responderam: gfs_sclose() • N_CONFIRM = 0 • N_RETRANS = 0 • oper = CLOSING_FILE • RESP := 0	Fechando
	confirm_reg_status(OK)	• Responde ao Usuário: confirm_gfs_open(fid)	Aberto
	confirm_reg_status(ERRO)	• Responde ao Usuário: confirm_gfs_open(-1) • N_CONFIRM = 0 • N_RETRANS = 0 • oper = CLOSING_FILE • Envia pedido aos servidores para que fechem seus segmentos locais: gfs_sclose(servidores) • RESP = 0 • Inicia *time-out*	Fechando
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = REG_STATUS)	• N_RETRANS++ • Retransmite pedido ao metaservidor para registro de status de arquivo: reg_file_status(ARQ_ABERTO)	Abrindo
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = REG_STATUS)	• Responde ao Usuário: confirm_gfs_open(-1) • N_CONFIRM = 0 • N_RETRANS = 0 • oper = CLOSING_FILE • RESP = 0	Fechando
Aberto	gfs_close() (ESPERA = 1)	• N_CONFIRM = 0 • RESP = 1 • oper = REG_CLOSING_FILE • Solicita ao metaservidor para fechar arquivo: reg_file_status(ARQ_FECHANDO) • Inicia *time-out*	Fechando

Aberto	gfs_close() (ESPERA = 0)	• Responde ao Usuário: confirm_gfs_close(fid) • N_CONFIRM = 0 • RESP = 0 • oper = REG_CLOSING_FILE • Solicita ao metaservidor para fechar arquivo: reg_file_status(ARQ_FECHANDO) • Inicia *time-out*	Fechando
	gfs_read	• Determina unidades de distribuição envolvidas: N_BLOCK = X • Envia pedido aos servidores: gfs_sread • N_CONFIRM = 0 • N_BYTES = 0 • N_RETRANS = 0 • Inicia *time-out*	Lendo
	gfs_write	• Determina unidades de distribuição envolvidas: N_BLOCK = X • Envia pedido aos servidores: gfs_swrite • N_CONFIRM = 0 • N_BYTES = 0 • N_RETRANS = 0 • Inicia *time-out*	Escrevendo
	gfs_lseek	• Ajusta posição corrente do arquivo • Responde ao Usuário: confirm_gfs_lseek	Aberto
Lendo	confirmdado (N_CONFIRM+1 < N_BLOCK)	• Copia dados do buffer • Atualiza confirmação do bloco • N_BYTES += BYTES_RECEBIDOS • N_CONFIRM++	Lendo
	confirmdado (N_CONFIRM+1 = N_BLOCK)	• Copia dados do buffer • N_BYTES += BYTES_RECEBIDOS • Return(N_BYTES)	Aberto
	Timeout (N_CONFIRM+1 < N_BLOCK)	• N_RETRANS++ • Retransmite pedido aos servidores que não responderam:gfs_sread()	Lendo
	Timeout (N_CONFIRM+1 = N_BLOCK)	• Return(N_BYTES)	Aberto
Escrevendo	confirmescrita (N_CONFIRM+1 < N_BLOCK)	• Atualiza confirmação do bloco • N_BYTES += BYTES_RECEBIDOS • N_CONFIRM++	Escrevendo
	confirmescrita (N_CONFIRM+1 = N_BLOCK)	• N_BYTES += BYTES_RECEBIDOS • Return(N_BYTES)	Aberto
	Timeout (N_CONFIRM+1 < N_BLOCK)	• N_RETRANS++ • Retransmite pedido aos servidores que não responderam:gfs_write()	Escrevendo
	Timeout (N_CONFIRM+1 = N_BLOCK)	• Return(N_BYTES)	Aberto
Fechando	confirm_reg_status (ARQ_FECHADO) (oper = REG_CLOSING_FILE)	• Solicita aos servidores envolvidos que fechem seus fragmentos locais: gfs_sclose(servidores) • N_CONFIRM = 0 • oper = CLOSING_FILE • Inicia *time-out*	Fechando
	confirm_reg_status(ERRO) (oper = REG_CLOSING_FILE && RESP = 1)	• Responde ao Usuário: confirm_gfs_close(status) • Cancela *time-out*	Espera
	confirm_reg_status(ERRO) (oper = REG_CLOSING_FILE && RESP = 0)	• Cancela *time-out*	Espera

Fechando	***Timeout*** (N_RETRANS < MAX_RETRANS) && (oper = REG_CLOSING_FILE)	• N_RETRANS++ • Retransmite envio de pedido de registro ao metaservidor: reg_file_status(ARQ_FECHANDO)	Fechando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = REG_CLOSING_FILE)	• Responde ao Usuário: confirm_gfs_close(-1)	Espera
	confirm_sclose() (N_CONFIRM+1 < N_SERVERS)	• Atualiza confirmação do servidor • N_CONFIRM++	Fechando
	confirm_sclose() (N_CONFIRM+1 = N_SERVERS)	• N_RETRANS = 0 • oper = REG_FILE_CLOSED • Solicita ao metaservidor registro de status de arquivo: reg_file_status (ARQ_FECHADO) • Inicia *time-out*	Fechando
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = CLOSING_FILE)	• N_RETRANS++ • Retransmite pedido aos servidores que não responderam:gfs_sclose()	Fechando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = CLOSING_FILE)	• N_RETRANS = 0 • oper = REG_FILE_CLOSED • Envia pedido de registro ao metaservidor: reg_file_status(ARQ_FECHADO) • Inicia *time-out*	Fechando
	confirm_reg_status(Status) (oper = REG_FILE_CLOSED && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_close(status)	Espera
	confirm_reg_status(Status) (oper = REG_FILE_CLOSED && RESP = 0)	• Cancela *time-out*	Espera
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = REG_FILE_CLOSED)	• N_RETRANS++ • Retransmite envio de pedido de registro ao metaservidor: reg_file_status(ARQ_FECHADO) • Inicia *time-out*	Fechando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = CLOSING_FILE)	• Responde ao Usuário: confirm_gfs_close(-1)	Espera
Removendo	gfs_unlink (ESPERA = 1)	• N_RETRANS = 0 • RESP = 1 • oper = REG_UNLINKING • Registra operação no metaservidor: reg_file_status_name(ARQ_REMOVENDO) • Inicia *time-out*	Removendo
	gfs_unlink (ESPERA = 0)	• Responde ao Usuário: confirm_gfs_unlink(OK) • N_RETRANS = 0 • RESP = 0 • oper = REG_UNLINKING • Registra operação no metaservidor: reg_file_status_name(ARQ_REMOVENDO) • Inicia *time-out*	Removendo
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = REG_UNLINKING)	• N_RETRANS++ • Retransmite registra de operação no metaservidor: reg_file_status_name(ARQ_REMOVENDO)	Removendo
	Time-out (N_RETRANS < MAX_RETRANS && oper = REG_UNLINKING && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_unlink(ERRO)	Espera

Removendo	*Timeout* (N_RETRANS < MAX_RETRANS && oper = REG_UNLINKING && RESP = 0)	• Cancela *time-out*	Espera
	confirm_reg_arq(OK) (oper = REG_UNLINKING)	• N_CONFIRM = 0 • N_RETRANS = 0 • oper = UNLINKING • N_SERVERS = X • Envia solicitação de remoção de arquivo aos servidores envolvidos: gfs_sunlink • Inicia *time-out*	Removendo
	confirm_reg_arq(ERRO) (oper = REG_UNLINKING && RESP =1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_unlink(ERRO)	Espera
	confirm_reg_arq(ERRO) (oper = REG_UNLINKING && RESP = 0)	• Cancela *time-out*	Espera
	confirm (N_CONFIRM+1 < N_SERVERS)	• Atualiza confirmação do servidor • CONFIRM++	Removendo
	confirm (N_CONFIRM+1 = N_SERVERS)	• N_RETRANS = 0 • oper = REG_UNLINKING_F • Registra término da operação no metaservidor: reg_file_status_name(ARQ_REMOVIDO)	Removendo
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = UNLINKING)	• N_RETRANS++ • Retransmite pedido aos servidores que não responderam: gfs_sunlink	Removendo
	Timeout (N_RETRANS < MAX_RETRANS && oper = UNLINKING && RESP = 1)	• Responde ao Usuário: confirm_gfs_unlink(ERRO) • oper = REG_UNLINKING_F • Registra término da operação no metaservidor: reg_file_status_name(ARQ_REMOVIDO)	Removendo
	Timeout (N_RETRANS < MAX_RETRANS && oper = UNLINKING && RESP = 0)	• oper = REG_UNLINKING_F • Registra término da operação no metaservidor: reg_file_status_name(ARQ_REMOVIDO)	Removendo
	confirm_reg_arq(status) (RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_unlink(status)	Espera
	confirm_reg_arq(status) (RESP = 0)	• Cancela *time-out*	Espera
	Timeout (N_RETRANS < MAX_RETRANS && oper = REG_UNLINKING_F)	• N_RETRANS++ • Retransmite registro de término da operação no metaservidor: reg_file_status_name(ARQ_REMOVIDO)	Removendo
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_UNLINKING_F && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_unlink(ERRO)	Espera
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_UNLINKING_F && RESP = 1)	• Cancela *time-out*	Espera

Renomeando	gfs_rename (ESPERA = 1)	• N_RETRANS = 0 • RESP = 1 • oper = REG_RENAMING • Registra operação no metaservidor: reg_file_status_name(ARQ_RENOMEANDO) • Inicia *time-out*	Renomeando
	gfs_rename (ESPERA = 0)	• Responde ao Usuário: confirm_gfs_rename(OK) • N_RETRANS = 0 • RESP = 0 • oper = REG_RENAMING • Registra operação no metaservidor: reg_file_status_name(ARQ_RENOMEANDO) • Inicia *time-out*	Renomeando
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = REG_RENAMING)	• N_RETRANS++ • Retransmite registro de operação no metaservidor: reg_file_status_name(ARQ_RENOMEANDO)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_RENAMING && RESP = 1)	• Cancela *time-out* • Responde ao Usuário:confirm_gfs_rename(ERRO)	Espera
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_RENAMING && RESP = 0)	• Cancela *time-out*	Espera
	confirm_reg_arq(OK)	• N_CONFIRM = 0 • N_SERVERS = X • oper = CONTACTING_SEC_MS • Registra operação no segundo metaservidor: reg_file_status_ren() • Inicia *time-out*	Renomeando
	confirm_reg_arq(ERRO)	• Responde ao Usuário:confirm_gfs_rename(ERRO)	Espera
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = REG_RENAMING)	• N_RETRANS++ • Retransmite registro de operação no segundo metaservidor: reg_file_status_ren()	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_RENAMING && RESP = 1)	• Responde ao Usuário: confirm_gfs_rename(ERRO) • N_RETRANS = 0 • oper = UN_REG_RENAMING • Restaura status do arquivo no metaservidor: reg_file_status_name(ARQ_FECHADO)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS && oper = REG_RENAMING && RESP = 0)	• N_RETRANS = 0 • oper = UN_REG_RENAMING • Restaura status do arquivo no metaservidor: reg_file_status_name(ARQ_FECHADO)	Renomeando

Renomeando	confirm_reg_na(OK)	• N_CONFIRM = 0 • N_RETRANS = 0 • oper = RENAMING • Entra em contato com os servidores envolvidos para que renomeiem seus segmentos locais: gfs_srename(servidores) • Inicia *Time-out*	Renomeando
	confirm_reg_na(ERRO)	• N_CONFIRM = 0 • oper = UN_REG_RENAMING • Restaura status do arquivo no metaservidor: reg_file_status_name(ARQ_FECHADO)	Renomeando
	Timeout *(N_RETRANS < MAX_RETRANS) && (oper = UM_REG_RENAMING)*	• N_RETRANS++ • Retransmite restauração do status do arquivo no metaservidor: reg_file_status_name(ARQ_FECHADO)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS && oper = UN_REG_RENAMING && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_rename(ERRO)	Espera
	Timeout (N_RETRANS = MAX_RETRANS && oper = UN_REG_RENAMING && RESP = 0)	• Cancela *time-out*	Espera
	confirm_reg_arq (oper = UN_REG_RENAMING && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_rename(ERRO)	Espera
	confirm_reg_arq (oper = UN_REG_RENAMING && RESP = 0)	• Cancela *time-out*	Espera
	confirm(status) (N_CONFIRM+1 < N_SERVERS)	• N_CONFIRM++ • Erro_oper = status • Reinicia *Time-out*	Renomeando
	confirm(status) (N_CONFIRM+1 = N_SERVERS) && (Erro_oper)	• N_RETRANS = 0 • oper = UN_REG_RENAMING_2 • Elimina arquivo que estava sendo criado no segundo metaservidor: reg_file_status_name(ARQ_REMOVIDO) • Inicia *Time-out*	Renomeando
	confirm(OK) (N_CONFIRM+1 = N_SERVERS) && (!Erro_oper)	• N_RETRANS = 0 • oper = CONFIRM_META_2 • Confirma êxito da operação ao segundo metaservidor: reg_file_status_rename(ARQ_RENOMEADO) • Inicia *Time-out*	Renomeando
	confirm(ERRO) (N_CONFIRM+1 = N_SERVERS) && (!Erro_oper)	• N_RETRANS = 0 • oper = UN_REG_RENAMING_2 • Elimina arquivo que estava sendo criado no segundo metaservidor: reg_file_status_name(ARQ_REMOVIDO) • Inicia *Time-out*	Renomeando
	confirm_reg_arq(status) (oper = N_REG_RENAMING_2)	• N_RETRANS = 0 • oper = UN_REG_RENAMING • Solicita alteração de status ao metaservidor: reg_file_status_name(ARQ_FECHADO) • Inicia *Time-out*	Renomeando

Renomeando	*Timeout* (N_RETRANS < MAX_RETRANS) && (oper = UN_REG_RENAMING2)	• N_RETRANS++ • Retransmite pedido ao segundo metaservidor para registrar fechamento de arquivo: reg_file_status_name(ARQ_FECHADO)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = UN_REG_RENAMING_2)	• N_RETRANS = 0 • oper = UN_REG_RENAMING • Solicita alteração de status ao metaservidor: reg_file_status_name(ARQ_FECHADO) • Inicia *Time-out*	Renomeando
	confirm_reg_arq(status) (oper = CONFIRM_META_2)	• N_RETRANS = 0 • oper = CONFIRM_META_1 • erro_oper = status • Confirma ao metaservidor renomeação de arquivo: reg_file_status_rename(RENOMEADO_M1) • Inicia *Time-out*	Renomeando
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = CONFIRM_META_2)	• N_RETRANS++ • Retransmite pedido ao segundo metaservidor para registrar renomeação de arquivo): reg_file_status_rename(RENOMEADO_M2)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = CONFIRM_META_2)	• N_RETRANS = 0 • oper = CONFIRM_META_1 • erro_oper = true • Solicita alteração de status ao metaservidor: reg_file_status_name(ARQ_FECHADO) • Inicia *Time-out*	Renomeando
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = CONFIRM_META_1)	• N_RETRANS++ • Retransmite ao metaservidor: reg_file_status_rename(RENOMEADO_M1)	Renomeando
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = CONFIRM_META_1 && RESP = 1)	• Cancela *time-out* • Responde ao Usuário: confirm_gfs_rename(ERRO)	Espera
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = CONFIRM_META_1 && RESP = 0)	• Cancela *time-out*	Espera
	confirm_reg_arq (oper = CONFIRM_META_1)	• Status = erro_oper • Responde ao Usuário: confirm_gfs_rename(status)	Espera
	Timeout (N_RETRANS < MAX_RETRANS) && (oper = RENAMING)	• N_RETRANS++ • Retransmite solicitação de renomeação de arquivo aos servidores que não responderam: gfs_srename(servidores)	Renomenado
	Timeout (N_RETRANS = MAX_RETRANS) && (oper = RENAMING)	• oper = UN_REG_RENAMING_2 • Contacta segundo metaservidor para desfazer a operação: reg_file_status_name(ARQ_FECHADO)	Renomeando

A.2 Diagrama de Transição de Estados do Processo Servidor

Estado Atual	Evento	Ação	Próximo Estado
Espera	gfs_open ()	• Abre/Cria segmentos locais • Responde ao Mestre: confirm_sopen (status)	Aberto
	gfs_unlink()	• Remove segmentos locais • Responde ao Mestre: confirm(status)	Espera
	gfs_rename()	• Renomeia segmentos locais • Responde ao Mestre: confirm(status)	Espera
Aberto	gfs_read()	• Lê bloco de dados solicitado • Envia dados para o cliente: confirm_dado(dados)	Aberto
	gfs_write	• Escreve bloco de dados recebido: confirm_escrita (status)	Aberto
	gfs_close()	• Fecha segmentos • Responde ao Mestre: confirm_sclose (status)	Espera

A.3 Diagrama de Transição de Estados do Meta-servidor

Estado Atual	*Evento*	Ação	Próximo Estado
Controlando	reg_file(fid)	• Registra Início de Operação na Tabela de Metadados • confirm_reg_file_serv(status)	Controlando
	reg_file_status_serv()	• Registra Finalização da Operação na Tabela de Metadados • confirm_reg_status_serv(status)	Controlando
	reg_file_status_name (arquivo)	• Registra situação atual do arquivo na Tabela de Metadados • confirm_reg_arq(status)	Controlando
	reg_file_status_rename (fid, arq)	• Registra operação de renomeação de arquivo em andamento • confirm_reg_arq(status)	Controlando
	reg_file_status_ren()	• Registra encerramento de operação de renomeação de arquivo • confirm_reg_na(status)	Controlando

Anexo B: Cenários para a simulação do Sistema de Arquivos Descentralizado

B.1 Cenários para a operação de Abertura de Arquivo

Operações Abertura de Arquivo bem-sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta METAServidor para registrar início de operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_open[S]

S2_Cli: Cliente retransmite pedido de registro de início de operação para o METAServidor até conseguir resposta
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) _(ABRINDO) reg_file[M]→ ... → (ABRINDO) _(ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_open[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M]
S4_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S5_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M] → (ABRINDO) confirm_reg_status_OK[M](ABERTO) confirm_gfs_open_OK [U]
S6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M] → (ABRINDO)_ (ABRINDO) reg_file_status [M] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [M] →(ABRINDO) confirm_reg_status_OK[M](ABERTO) confirm_gfs_open_OK [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor registra início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_OK[C]
S2_M: Meta-Servidor registra término de operação de abertura de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor abre arquivo com sucesso
(ESPERA)gfs_sopen[C](ABERTO) confirm_sopen_OK[C]

Cenários que levam à abertura de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 2: (S1_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 3: (S1_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 4: (S1_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 5: (S2_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv

Cenário 6: (S2_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 7: (S2_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 8: (S2_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv

Operação de Abertura de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_ERRO[M](ESPERA) confirm_gfs_open_ERRO [U]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Meta-Servidor retorna ERRO para a operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) _(ABRINDO) reg_file[M]→ ... → (ABRINDO) _(ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_ERRO[M](ESPERA) confirm_gfs_open_ERRO [U]
S3_Cli: Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) _(ABRINDO) reg_file[M]→ ... → (ABRINDO) _(ABRINDO) reg_file[M] → (ABRINDO)_ (ESPERA) confirm_gfs_open_ERRO [U]
S4_Cli: Cliente contacta com sucesso Meta-Servidor para registrar início de operação
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sclose[S]
S5_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_open[U](ABRINDO) reg_file[M] → (ABRINDO) _(ABRINDO) reg_file[M]→ ... → (ABRINDO) _(ABRINDO) reg_file[M] → (ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sclose[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta-Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_ERRO[S](ABRINDO) confirm_gfs_open_ERRO [U], reg_file_status[M] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
S7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta-Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_ERRO[S](ABRINDO) confirm_gfs_open_ERRO [U] , reg_file_status[M] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta. Meta-Servidor é avisado sobre a falha da operação
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) _(ABRINDO) confirm_gfs_open_ERRO [U] , reg_file_status[M] → (ABRINDO) confirm_reg_status_OK(ESPERA)_
S9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M]
S10_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(ABRINDO) confirm_reg_status_OK[M](ABRINDO) gfs_sopen[S] → (ABRINDO) _(ABRINDO) gfs_sopen[S]→...→ (ABRINDO) _(ABRINDO) gfs_sopen[S] →(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S11_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M] → (ABRINDO) confirm_reg_status_ERRO[S](ABERTO) confirm_gfs_open_ERRO [U]

S12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M] → (ABRINDO)_ (ABRINDO) reg_file_status [M] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [M] →(ABRINDO) confirm_reg_status_ERRO[S](ABERTO) confirm_gfs_open_ERRO [U]
S13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(ABRINDO) confirm_sopen_OK[S](ABRINDO) reg_file_status [M] → (ABRINDO)_ (ABRINDO) reg_file_status [M] →...→ (ABRINDO)_ (ABRINDO) reg_file_status [M] →(ABRINDO)_(ESPERA) confirm_gfs_open_ERRO [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor registra com sucesso início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_OK[C]
S2_M: Meta-Servidor registra com erro início de operação de abertura de arquivo
(CONTROLANDO) reg_file[C](CONTROLANDO) confirm_reg_status_ERRO[C]
S3_M: Meta-Servidor registra término de operação de abertura de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor abre arquivo com sucesso
(ESPERA)gfs_sopen[C](ABERTO) confirm_sopen_OK[C]
S2_Serv: Servidor falha ao abrir arquivo
(ESPERA)gfs_sopen[C](ESPERA) confirm_sopen_ERRO[C]

Cenários que levam à abertura de arquivo com sucesso

Cenário 1: (S1_Cli)
Cenário 2: (S2_Cli)
Cenário 3: (S3_Cli)
Cenário 4: (S4_Cli ; S6_Cli) → (S3_M) → (S2_Serv)
Cenário 5: (S4_Cli ; S7_Cli) → (S3_M) → (S2_Serv)
Cenário 6: (S4_Cli ; S8_Cli) → (S3_M)
Cenário 7: (S4_Cli ; S9_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 8: (S4_Cli ; S9_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 9: (S4_Cli ; S9_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 10: (S4_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 11: (S4_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 12: (S4_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 13: (S5_Cli ; S6_Cli) → (S2_M) → (S2_Serv)
Cenário 14: (S5_Cli ; S7_Cli) → (S2_M) → (S2_Serv)
Cenário 15: (S5_Cli ; S8_Cli) → (S2_M)
Cenário 16: (S5_Cli ; S9_Cli; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 17: (S5_Cli ; S9_Cli; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 18: (S5_Cli ; S9_Cli;;S13_Cli) → (S1_M) → (S1_Serv)
Cenário 19: (S5_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 20: (S5_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 21: (S5_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)

B.2 Cenários para a operação de Fechamento de Arquivo

Operações de Fechamento de Arquivo bem sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]
S4_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] →(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S5_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](ESPERA) confirm_gfs_close_OK [U]
S6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](ESPERA) confirm_gfs_close_OK [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor registra início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_OK[C]
S2_M: Meta-Servidor registra término de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor fecha arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_OK[C]

Cenários que levam ao fechamento de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 2: (S1_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 3: (S1_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 4: (S1_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 5: (S2_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 6: (S2_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 7: (S2_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 8: (S2_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv

Operações de Fechamento de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M] (ESPERA) confirm_gfs_close_ERRO [U]

S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Meta-Servidor retorna ERRO para a operação
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M] (ESPERA) confirm_gfs_close_ERRO [U]
S3_Cli: Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO)_[M] (ESPERA) confirm_gfs_close_ERRO [U]
S4_Cli: Cliente contacta Meta-Servidor com sucesso para registrar início de operação
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]
S5_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta-Servidor é avisado sobre a falha da operação
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_ERRO[S](FECHANDO) confirm_gfs_close_ERRO [U], reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)_
S7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta-Servidor é avisado sobre a falha da operação
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_ERRO[S](FECHANDO) confirm_gfs_close_ERRO [U], reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)_
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) confirm_gfs_close_ERRO [U], reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)_
S9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]
S10_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] →(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S11_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M](ESPERA) confirm_gfs_close_ERRO [U]
S12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M](ESPERA) confirm_gfs_close_ERRO [U]
S13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor. Falha na Rede impede contato.
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO)_[M](ESPERA) confirm_gfs_close_ERRO [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_ERRO[C]
S2_M: Meta-Servidor falha ao registrar término de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_ERRO[C]
S3_M: Meta-Servidor registra início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor falha ao fechar arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_ERRO[C]
S2_Serv: Servidor fecha arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_OK[C]

Cenários que levam ao fechamento de arquivo sem sucesso

Cenário 1: (S1_Cli)
Cenário 2: (S2_Cli)
Cenário 3: (S3_Cli)
Cenário 4: (S4_Cli ; S6_Cli) → (S3_M) → (S2_Serv)
Cenário 5: (S4_Cli ; S7_Cli) → (S3_M) → (S2_Serv)
Cenário 6: (S4_Cli ; S8_Cli) → (S3_M)
Cenário 7: (S4_Cli ; S9_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 8: (S4_Cli ; S9_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 9: (S4_Cli ; S9_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 10: (S4_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 11: (S4_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 12: (S4_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 13: (S5_Cli ; S6_Cli) → (S2_M) → (S2_Serv)
Cenário 14: (S5_Cli ; S7_Cli) → (S2_M) → (S2_Serv)
Cenário 15: (S5_Cli ; S8_Cli) → (S2_M)
Cenário 16: (S5_Cli ; S9_Cli; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 17: (S5_Cli ; S9_Cli; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 18: (S5_Cli ; S9_Cli;;S13_Cli) → (S1_M) → (S1_Serv)
Cenário 19: (S5_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 20: (S5_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 21: (S5_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)

Operações Assíncronas de Fechamento de Arquivo bem sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]
S4_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta

(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] →(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S5_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](ESPERA)_
S6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](ESPERA)_

Situações no Meta-Servidor

S1_M: Meta-Servidor registra início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_OK[C]
S2_M: Meta-Servidor registra término de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor fecha arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_OK[C]

Cenários que levam ao fechamento de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 2: (S1_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 3: (S1_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 4: (S1_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 5: (S2_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 6: (S2_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 7: (S2_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 8: (S2_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv

Operações Assíncronas de Fechamento de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente recebe confirmação de realização de operação de fechamento de arquivo. Em seguida, Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) confirm_reg_status_ERRO[M] (ESPERA)_
S2_Cli: Cliente recebe confirmação de realização de operação de fechamento de arquivo. Em seguida, Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Meta-Servidor retorna ERRO para a operação
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M] (ESPERA)_
S3_Cli: Cliente recebe confirmação de realização de operação de fechamento de arquivo. Em seguida,, Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO)_[M] (ESPERA)_
S4_Cli: Cliente recebe confirmação de realização de operação de fechamento de arquivo. Em seguida, Cliente contacta Meta-Servidor com sucesso para registrar início de operação

(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]
S5_Cli: Cliente recebe confirmação de realização de operação de fechamento de arquivo. Em seguida, Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ABERTO)gfs_close [U](FECHANDO) reg_file_status [M], confirm_gfs_close_OK [U] → (FECHANDO) _(FECHANDO) reg_file_status [M]→ ... → (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S]]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta-Servidor é avisado sobre a falha da operação
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_ERRO[S](FECHANDO)reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)
S7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta-Servidor é avisado sobre a falha da operação
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_ERRO[S](FECHANDO) reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) reg_file_status[M] → (FECHANDO)confirm_reg_status_OK[M](ESPERA)
S9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]
S10_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(FECHANDO) confirm_reg_status_OK[M](FECHANDO) gfs_sclose[S] → (FECHANDO) _(FECHANDO) gfs_sclose[S]→...→ (FECHANDO) _(FECHANDO) gfs_sclose[S] →(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S11_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M](ESPERA)_
S12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO)_(FECHANDO) reg_file_status [M] → (FECHANDO) confirm_reg_status_ERRO[M](ESPERA)_
S13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor. Falha na Rede impede contato.
(FECHANDO) confirm_sclose_OK[S](FECHANDO) reg_file_status [M] → (FECHANDO) _(FECHANDO) reg_file_status [M] →...→ (FECHANDO) _(FECHANDO) reg_file_status [M] → (FECHANDO)_[M](ESPERA)_

Situações no Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_ERRO[C]
S2_M: Meta-Servidor falha ao registrar término de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status [C](CONTROLANDO) confirm_reg_status_ERRO[C]
S3_M: Meta-Servidor registra início de operação de fechamento de arquivo
(CONTROLANDO) reg_file_status[C](CONTROLANDO) confirm_reg_status_OK[C]

Situações no Servidor

S1_Serv: Servidor falha ao fechar arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_ERRO[C]
S2_Serv: Servidor fecha arquivo com sucesso
(ABERTO)gfs_close[C](ESPERA) confirm_sclose_OK[C]

Cenários que levam ao fechamento de arquivo sem sucesso

Cenário 1: (S1_Cli)
Cenário 2: (S2_Cli)
Cenário 3: (S3_Cli)
Cenário 4: (S4_Cli ; S6_Cli) → (S3_M) → (S2_Serv)
Cenário 5: (S4_Cli ; S7_Cli) → (S3_M) → (S2_Serv)
Cenário 6: (S4_Cli ; S8_Cli) → (S3_M)
Cenário 7: (S4_Cli ; S9_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 8: (S4_Cli ; S9_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 9: (S4_Cli ; S9_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 10: (S4_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 11: (S4_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 12: (S4_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 13: (S5_Cli ; S6_Cli) → (S2_M) → (S2_Serv)
Cenário 14: (S5_Cli ; S7_Cli) → (S2_M) → (S2_Serv)
Cenário 15: (S5_Cli ; S8_Cli) → (S2_M)
Cenário 16: (S5_Cli ; S9_Cli; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 17: (S5_Cli ; S9_Cli; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 18: (S5_Cli ; S9_Cli;;S13_Cli) → (S1_M) → (S1_Serv)
Cenário 19: (S5_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 20: (S5_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 21: (S5_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)

B.3 Cenários para a operação de Remoção de Arquivo

Operações de Remoção de Arquivo bem sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]
S4_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S5_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq _OK[M](ESPERA) confirm_gfs_unlink_OK [U]
S6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq _OK[M](ESPERA) confirm_gfs_unlink_OK [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor registra início de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_OK[C]
S2_M: Meta-Servidor registra término de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name [C](CONTROLANDO) confirm_reg_arq_OK[C]

Situações no Servidor

S1_Serv: Servidor remove arquivo com sucesso
(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_OK[C]

Cenários que levam ao fechamento de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 2: (S1_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 3: (S1_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 4: (S1_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 5: (S2_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 6: (S2_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 7: (S2_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 8: (S2_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv

Operações de Remoção de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA) confirm_gfs_unlink_ERRO [U]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Meta-Servidor retorna ERRO para a operação
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA) confirm_gfs_unlink_ERRO [U]
S3_Cli: Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO)_(ESPERA) confirm_gfs_unlink_ERRO [U]
S4_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]
S5_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta-Servidor é avisado sobre a falha da operação
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_ERRO[S] (FECHANDO) confirm_gfs_unlink_ERRO [U], reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)_
S7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta-Servidor é avisado sobre a falha da operação
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→...→ (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_ERRO[S] (FECHANDO) confirm_gfs_unlink_ERRO [U], reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)_
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→...→ (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(FECHANDO) confirm_gfs_unlink_ERRO [U], reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)_
S9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]
S10_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S11_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA) confirm_gfs_unlink_ERRO [U]
S12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] →...→ (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA) confirm_gfs_unlink_ERRO [U]
S13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor. Falha na Rede impede contato.
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] →...→ (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO)_(ESPERA) confirm_gfs_unlink_ERRO [U]

Situações no Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_ERRO[C]
S2_M: Meta-Servidor falha ao registrar término de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_ERRO[C]
S3_M: Meta-Servidor registra início de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_OK[C]

Situações no Servidor

S1_Serv: Servidor falha ao remover arquivo com sucesso
(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_ERRO[C]

S2_Serv: Servidor remove arquivo com sucesso
(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_OK[C]

Cenários que levam à remoção de arquivo sem sucesso

Cenário 1: (S1_Cli)
Cenário 2: (S2_Cli)
Cenário 3: (S3_Cli)
Cenário 4: (S4_Cli ; S6_Cli) → (S3_M) → (S2_Serv)
Cenário 5: (S4_Cli ; S7_Cli) → (S3_M) → (S2_Serv)
Cenário 6: (S4_Cli ; S8_Cli) → (S3_M)
Cenário 7: (S4_Cli ; S9_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 8: (S4_Cli ; S9_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 9: (S4_Cli ; S9_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 10: (S4_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 11: (S4_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 12: (S4_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 13: (S5_Cli ; S6_Cli) → (S2_M) → (S2_Serv)
Cenário 14: (S5_Cli ; S7_Cli) → (S2_M) → (S2_Serv)
Cenário 15: (S5_Cli ; S8_Cli) → (S2_M)
Cenário 16: (S5_Cli ; S9_Cli; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 17: (S5_Cli ; S9_Cli; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 18: (S5_Cli ; S9_Cli;;S13_Cli) → (S1_M) → (S1_Serv)
Cenário 19: (S5_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 20: (S5_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 21: (S5_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)

Operações Assíncronas de Remoção de Arquivo bem sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M], confirm_gfs_unlink_OK [U] → (REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M], confirm_gfs_unlink_OK [U] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S3_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]
S4_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S5_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq _OK[M](ESPERA)_
S6_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta

(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq _OK[M](ESPERA)_

Situações no Meta-Servidor

S1_M: Meta-Servidor registra início de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_OK[C]
S2_M: Meta-Servidor registra término de operação de remoção de arquivo
(CONTROLANDO) reg_file_status_name [C](CONTROLANDO) confirm_reg_arq_OK[C]

Situações no Servidor

S1_Serv: Servidor remove arquivo com sucesso
(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_OK[C]

Cenários que levam ao fechamento de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 2: (S1_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 3: (S1_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 4: (S1_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 5: (S2_Cli ; S3_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 6: (S2_Cli ; S4_Cli ; S5_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 7: (S2_Cli ; S3_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv
Cenário 8: (S2_Cli ; S4_Cli ; S6_Cli) → (S1_M ; S2_M) → S1_Serv

Operações Assíncronas de Remoção de Arquivo mal sucedidas

Situações no Cliente:

Cliente tenta contactar Meta-Servidor para registrar início da operação

S1_Cli: Cliente recebe confirmação de realização de operação de remoção de arquivo. Em seguida, Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M], confirm_gfs_unlink_OK [U] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA)_
S2_Cli: Cliente recebe confirmação de realização de operação de remoção de arquivo. Em seguida, Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Meta-Servidor retorna ERRO para a operação
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA)
S3_Cli: Cliente recebe confirmação de realização de operação de remoção de arquivo. Em seguida, Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ESPERA)gfs_unlink [U](REMOVENDO) reg_file_status_name [M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M]→ ... → (REMOVENDO) _(REMOVENDO) reg_file_status_name [M] → (REMOVENDO)_(ESPERA)_
S4_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]
S5_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta
(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S6_Cli: Cliente contacta Servidor envolvido com o arquivo distribuído,que retorna ERRO. Meta-Servidor é avisado sobre a falha da operação

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_ERRO[S] (FECHANDO) reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)

S7_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor até conseguir ERRO como resposta. Meta-Servidor é avisado sobre a falha da operação

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→...→ (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_ERRO[S] (FECHANDO) reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)

S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→...→ (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(FECHANDO) reg_file_status_name[M] → (REMOVENDO)confirm_reg_arq_OK[M](ESPERA)_

S9_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](RENOMEANDO) reg_file_status_name[M]

S10_Cli: Cliente retransmite pedido de fechamento de arquivo aos servidores até conseguir resposta

(REMOVENDO) confirm_reg_arq_OK[M](REMOVENDO) gfs_sunlink[S] → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S]→ ... → (REMOVENDO) _(REMOVENDO) gfs_sunlink[S] → (REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M]

Cliente tenta contactar Meta-Servidor para registrar término da operação

S11_Cli: Cliente contacta Meta-Servidor para registrar término de operação

(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA)_

S12_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta

(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] →...→ (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO) confirm_reg_arq_ERRO[M](ESPERA)_

S13_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor. Falha na Rede impede contato.

(REMOVENDO) confirm_OK[S](REMOVENDO) reg_file_status_name[M] → (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] →...→ (REMOVENDO) _(REMOVENDO) reg_file_status_name[M] → (REMOVENDO)_(ESPERA)_

Situações no Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de remoção de arquivo

(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_ERRO[C]

S2_M: Meta-Servidor falha ao registrar término de operação de remoção de arquivo

(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_ERRO[C]

S3_M: Meta-Servidor registra início de operação de remoção de arquivo

(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq_OK[C]

Situações no Servidor

S1_Serv: Servidor falha ao remover arquivo com sucesso

(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_ERRO[C]

S2_Serv: Servidor remove arquivo com sucesso

(ESPERA)gfs_unlink[C](ESPERA) confirm_unlink_OK[C]

Cenários que levam à remoção de arquivo sem sucesso

Cenário 1: (S1_Cli)
Cenário 2: (S2_Cli)
Cenário 3: (S3_Cli)
Cenário 4: (S4_Cli ; S6_Cli) → (S3_M) → (S2_Serv)
Cenário 5: (S4_Cli ; S7_Cli) → (S3_M) → (S2_Serv)

Cenário 6: (S4_Cli ; S8_Cli) → (S3_M)
Cenário 7: (S4_Cli ; S9_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 8: (S4_Cli ; S9_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 9: (S4_Cli ; S9_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 10: (S4_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 11: (S4_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 12: (S4_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)
Cenário 13: (S5_Cli ; S6_Cli) → (S2_M) → (S2_Serv)
Cenário 14: (S5_Cli ; S7_Cli) → (S2_M) → (S2_Serv)
Cenário 15: (S5_Cli ; S8_Cli) → (S2_M)
Cenário 16: (S5_Cli ; S9_Cli; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 17: (S5_Cli ; S9_Cli; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 18: (S5_Cli ; S9_Cli;;S13_Cli) → (S1_M) → (S1_Serv)
Cenário 19: (S5_Cli ; S10_Cli ; S11_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 20: (S5_Cli ; S10_Cli ; S12_Cli) → (S1_M ; S3_M) → (S1_Serv)
Cenário 21: (S5_Cli ; S10_Cli ; S13_Cli) → (S1_M) → (S1_Serv)

B.4 Cenários para a operação de Renomeação de Arquivo

Operações de Renomeação de Arquivo bem sucedidas

Situações no Cliente:

1. Cliente tenta contactar Primeiro Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]

2. Cliente tenta contactar Segundo Meta-Servidor para registrar início da operação

S3_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]
S4_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

3. Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S5_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]
S6_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]

4. Cliente tenta contactar Segundo Meta-Servidor para registrar término da operação

S7_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]
S8_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta

(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]

5. Cliente tenta contactar Primeiro Meta-Servidor para registrar término da operação

S9_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_OK[U]
S10_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)
confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_OK[U]

Situações no Primeiro Meta-Servidor

S1_M: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]
S2_M: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]

Situações no Segundo Meta-Servidor

S1_M2: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_OK[C]
S2_M2: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _OK [C]

Situações no Servidor

S1_Serv: Servidor renomeia_arquivo com sucesso
(ESPERA)gfs_srename[C](ABERTO) confirm_OK[C]

Cenários que levam à renomeação de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli; S5_Cli; S7_Cli ; S9_Cli) → (S1_M; S2_M; S1_M; S2_M) → (S1_Serv)
Cenário 2: (S1_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 3: (S1_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 4: (S1_Cli ; S3_Cli ; S5_Cli; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 5: (S1_Cli ; S3_Cli ; S6_Cli; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 6: (S1_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 7: (S1_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 8: (S1_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 9: (S1_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 10: (S1_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 11: (S1_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 12: (S1_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 13: (S1_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 14: (S1_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 15: (S1_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 16: (S1_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 17: (S2_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 18: (S2_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 19: (S2_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 20: (S2_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 21: (S2_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 22: (S2_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 23: (S2_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)

Cenário 24: (S2_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 25: (S2_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 26: (S2_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 27: (S2_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 28: (S2_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 29: (S2_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 30: (S2_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 31: (S2_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 32: (S2_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)

Operações de Renomeação de Arquivo mal sucedidas

Situações no Cliente:

1. Cliente tenta contactar Primeiro Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)confirm_gfs_rename_ERRO[U]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)confirm_gfs_rename_ERRO[U
S3_Cli: Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)_(ESPERA)confirm_gfs_rename_ERRO[U]
S4_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]
S5_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]

2. Cliente tenta contactar Segundo Meta-Servidor para registrar início da operação

S6_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)confirm_gfs_rename_ERRO[U]
S7_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)confirm_gfs_rename_ERRO[U
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO)_(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)confirm_gfs_rename_ERRO[U]
S9_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

S10_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

3. Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S11_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído. Erro retornado pelo servidor leva Cliente a desfazer o registro de início de operação nos dois M servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_ERRO[S](RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S12_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta. Erro retornado pelo servidor leva Cliente a desfazer o registro de início de operação nos dois M servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_ERRO[S](RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S13_Cli: Cliente retransmite pedido de renomeação de arquivo ao servidor, mas não consegue resposta. Cliente desfaz o registro de início de operação nos dois M servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO)_(RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S14_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]
S15_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]

4. Cliente tenta contactar Segundo Meta-Servidor para registrar término da operação

S16_Cli: Cliente contacta Meta-Servidor para registrar início de operação. Erro leva o Cliente a anular operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S17_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Erro leva o Cliente a anular operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S18_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor, mas não consegue resposta. Cliente anula o registro de operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO)_[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)confirm_gfs_rename_ERRO[U]
S19_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]
S20_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta

(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]

5. Cliente tenta contactar Primeiro Meta-Servidor para registrar término da operação

S21_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(RENOMEANDO) confirm_reg_arq_OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)confirm_gfs_rename_ERRO[U]
S22_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)
confirm_reg_arq_ERRO[M](ESPERA)confirm_gfs_rename_ERRO[U]
S23_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor, mas não consegue resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)_(ESPERA)confirm_gfs_rename_ERRO[U]

Situações no Primeiro Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _ERRO[C]
S2_M: Meta-Servidor falha ao registrar término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _ERRO[C]
S3_M: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]

Situações no Segundo Meta-Servidor

S1_M2: Meta-Servidor falha ao registrar início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_ERRO[C]
S2_M2: Meta-Servidor falha ao registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _ERRO [C]
S3_M2: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_OK[C]
S4_M2: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _OK [C]

Situações no Servidor

S1_Serv: Servidor renomeia arquivo com sucesso
(ESPERA)gfs_srename[C](ABERTO) confirm_OK[C]
S2_Serv: Servidor falha ao renomear arquivo
(ESPERA)gfs_srename[C](ABERTO) confirm_ERRO[C]

Cenários que levam à renomeação de arquivo mal-sucedidas

Cenário 1: (S1_Cli) → (S1_M)
Cenário 2: (S2_Cli) → (S1_M)
Cenário 3: (S3_Cli) → (S1_M)
Cenário 4: (S4_Cli; S6_Cli) → (S3_M) → (S1_M2)
Cenário 5: (S4_Cli; S7_Cli) → (S3_M) → (S1_M2)
Cenário 6: (S4_Cli; S8_Cli) → (S3_M) → (S1_M2)
Cenário 7: (S4_Cli; S9_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 8: (S4_Cli; S9_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 9: (S4_Cli; S9_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)

Cenário 10: (S4_Cli; S9_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 11: (S4_Cli; S9_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 12: (S4_Cli; S9_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 13: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 14: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 15: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 16: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 17: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 18: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 19: (S4_Cli; S9_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 20: (S4_Cli; S9_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 21: (S4_Cli; S9_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 22: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 23: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 24: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 25: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 26: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 27: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 28: (S4_Cli; S10_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 29: (S4_Cli; S10_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 30: (S4_Cli; S10_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 31: (S4_Cli; S10_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 32: (S4_Cli; S10_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 33 (S4_Cli; S10_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 34 (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 35 (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 36: (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 37: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 38: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 39: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 40: (S4_Cli; S10_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 41: (S4_Cli; S10_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 42: (S4_Cli; S10_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 43: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 44: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 45: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 46: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 47: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 48: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)

Cenário 49: (S4_Cli; S6_Cli) → (S3_M) → (S1_M2)
Cenário 50: (S5_Cli; S7_Cli) → (S3_M) → (S1_M2)
Cenário 51: (S5_Cli; S8_Cli) → (S3_M) → (S1_M2)
Cenário 52: (S5_Cli; S9_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 53: (S5_Cli; S9_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 54: (S5_Cli; S9_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 55: (S5_Cli; S9_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 56: (S5_Cli; S9_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 57: (S5_Cli; S9_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 58: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 59: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 60: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 61: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 62: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 63: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 64: (S5_Cli; S9_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 65: (S5_Cli; S9_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 66: (S5_Cli; S9_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 67: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 68: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)

Cenário 69: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 70: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 71: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 72: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 73: (S5_Cli; S10_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 74: (S5_Cli; S10_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 75: (S5_Cli; S10_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 76: (S5_Cli; S10_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 77: (S5_Cli; S10_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 78 (S5_Cli; S10_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 79 (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 80 (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 81: (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 82: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 83: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 84: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 85: (S5_Cli; S10_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 86: (S5_Cli; S10_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 87: (S5_Cli; S10_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 88: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 89: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 90: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 91: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 92: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 93: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)

Operações Assíncronas de Renomeação de Arquivo bem sucedidas

Situações no Cliente:

1. Cliente tenta contactar Primeiro Meta-Servidor para registrar início da operação

S1_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]
S2_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]

2. Cliente tenta contactar Segundo Meta-Servidor para registrar início da operação

S3_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]
S4_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

3. Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S5_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]
S6_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta

(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]

4. Cliente tenta contactar Segundo Meta-Servidor para registrar término da operação

S7_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]
S8_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]

5. Cliente tenta contactar Primeiro Meta-Servidor para registrar término da operação

S9_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S10_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_

Situações no Primeiro Meta-Servidor

S1_M: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]
S2_M: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]

Situações no Segundo Meta-Servidor

S1_M2: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_OK[C]
S2_M2: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _OK [C]

Situações no Servidor

S1_Serv: Servidor renomeia arquivo com sucesso
(ESPERA)gfs_srename[C](ABERTO) confirm_OK[C]

Cenários que levam à renomeação de arquivo com sucesso

Cenário 1: (S1_Cli ; S3_Cli; S5_Cli; S7_Cli ; S9_Cli) → (S1_M; S2_M; S1_M; S2_M) → (S1_Serv)
Cenário 2: (S1_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 3: (S1_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 4: (S1_Cli ; S3_Cli ; S5_Cli; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 5: (S1_Cli ; S3_Cli ; S6_Cli; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 6: (S1_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 7: (S1_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 8: (S1_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 9: (S1_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 10: (S1_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 11: (S1_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 12: (S1_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 13: (S1_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 14: (S1_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)

Cenário 15: (S1_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 16: (S1_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 17: (S2_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 18: (S2_Cli ; S3_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 19: (S2_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 20: (S2_Cli ; S3_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 21: (S2_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 22: (S2_Cli ; S3_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 23: (S2_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 24: (S2_Cli ; S3_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 25: (S2_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 26: (S2_Cli ; S4_Cli ; S5_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 27: (S2_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 28: (S2_Cli ; S4_Cli ; S5_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 29: (S2_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 30: (S2_Cli ; S4_Cli ; S6_Cli ; S7_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 31: (S2_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S9_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)
Cenário 32: (S2_Cli ; S4_Cli ; S6_Cli ; S8_Cli ; S10_Cli) → (S1_M ; S2_M ; S1_M ; S2_M) → (S1_Serv)

Operações Assíncronas de Renomeação de Arquivo mal sucedidas

Situações no Cliente:

1. Cliente tenta contactar Primeiro Meta-Servidor para registrar início da operação

S1_Cli: Cliente recebe confirmação de realização de operação de renomeação de arquivo. Em seguida, Cliente contacta Meta-Servidor para registrar início de operação, que retorna ERRO
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)_
S2_Cli: Cliente recebe confirmação de realização de operação de remoção de arquivo. Em seguida, Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir ERRO como resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)_
S3_Cli: Cliente recebe confirmação de realização de operação de remoção de arquivo. Em seguida, Cliente retransmite pedido, mas não consegue contactar o Meta-Servidor
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)_(ESPERA)_
S4_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]
S5_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(ESPERA) gfs_rename[U](RENOMEANDO) reg_file_status_name[M], confirm_gfs_rename_OK[U] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2]

2. Cliente tenta contactar Segundo Meta-Servidor para registrar início da operação

S6_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)_

S7_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO)

reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)_
S8_Cli: Cliente retransmite pedido de abertura de arquivo ao servidor, mas não consegue resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO)_(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)confirm_reg_arq_status[M] (ESPERA)_
S9_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

S10_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq_OK[M](RENOMEANDO) reg_file_status_ren[M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_ren[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M_2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S]

3. Cliente tenta contactar Servidores envolvidos com o arquivo distribuído

S11_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído. Erro retornado pelo servidor leva Cliente a desfazer o registro de início de operação nos dois Meta-servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_ERRO[S](RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S12_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta. Erro retornado pelo servidor leva Cliente a desfazer o registro de início de operação nos dois Meta-servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_ERRO[S](RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S13_Cli: Cliente retransmite pedido de renomeação de arquivo ao servidor, mas não consegue resposta. Cliente desfaz o registro de início de operação nos dois Meta-servidores
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO)_(RENOMEANDO) reg_file_status_name[M2] → (RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) reg_file_status_name[M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S14_Cli: Cliente contacta Servidores envolvidos com o arquivo distribuído
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]
S15_Cli: Cliente retransmite pedido de abertura de arquivo aos servidores até conseguir resposta
(RENOMEANDO) confirm_reg_na_OK[M_2](RENOMEANDO) gfs_srename[S] → (RENOMEANDO) _(RENOMEANDO) gfs_srename[S]→...→ (RENOMEANDO) _(RENOMEANDO) gfs_srename[S] →(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2]

4. Cliente tenta contactar Segundo Meta-Servidor para registrar término da operação

S16_Cli: Cliente contacta Meta-Servidor para registrar início de operação. Erro leva o Cliente a anular operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S17_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta. Erro leva o Cliente a anular operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _ERRO[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_

S18_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor, mas não consegue resposta. Cliente anula o registro de operação no Primeiro Meta-Servidor
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO)_[M_2](RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_OK[M](ESPERA)_
S19_Cli: Cliente contacta Meta-Servidor para registrar início de operação
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]
S20_Cli: Cliente retransmite pedido de registro de início de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_OK[S](RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename[M_2]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_rename [M_2] → (RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M]

5. Cliente tenta contactar Primeiro Meta-Servidor para registrar término da operação

S21_Cli: Cliente contacta Meta-Servidor para registrar término de operação
(RENOMEANDO) confirm_reg_arq_OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)_

S22_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor até conseguir resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO) confirm_reg_arq_ERRO[M](ESPERA)_

S23_Cli: Cliente retransmite pedido de registro de término de operação para o Meta-Servidor, mas não consegue resposta
(RENOMEANDO) confirm_reg_arq _OK[M_2](RENOMEANDO) reg_file_status_name [M] →
(RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M]→ ... → (RENOMEANDO) _(RENOMEANDO) reg_file_status_name [M] → (RENOMEANDO)_(ESPERA)_

Situações no Primeiro Meta-Servidor

S1_M: Meta-Servidor falha ao registrar início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _ERRO[C]
S2_M: Meta-Servidor falha ao registrar término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _ERRO[C]
S3_M: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_name[C](CONTROLANDO) confirm_reg_arq _OK[C]

Situações no Segundo Meta-Servidor

S1_M2: Meta-Servidor falha ao registrar início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_ERRO[C]
S2_M2: Meta-Servidor falha ao registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _ERRO [C]
S3_M2: Meta-Servidor registra início de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_ren [C](CONTROLANDO) confirm_reg_na_OK[C]
S4_M2: Meta-Servidor registra término de operação de renomeação de arquivo
(CONTROLANDO) reg_file_status_rename[C](CONTROLANDO) confirm_reg_arq _OK [C]

Situações no Servidor

S1_Serv: Servidor renomeia arquivo com sucesso
(ESPERA)gfs_srename[C](ABERTO) confirm_OK[C]
S2_Serv: Servidor falha ao renomear arquivo
(ESPERA)gfs_srename[C](ABERTO) confirm_ERRO[C]

Cenários que levam à renomeação de arquivo mal-sucedidas

Cenário 1: (S1_Cli) → (S1_M)
Cenário 2: (S2_Cli) → (S1_M)
Cenário 3: (S3_Cli) → (S1_M)
Cenário 4: (S4_Cli; S6_Cli) → (S3_M) → (S1_M2)
Cenário 5: (S4_Cli; S7_Cli) → (S3_M) → (S1_M2)
Cenário 6: (S4_Cli; S8_Cli) → (S3_M) → (S1_M2)
Cenário 7: (S4_Cli; S9_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 8: (S4_Cli; S9_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 9: (S4_Cli; S9_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 10: (S4_Cli; S9_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 11: (S4_Cli; S9_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 12: (S4_Cli; S9_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 13: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 14: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 15: (S4_Cli; S9_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 16: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 17: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 18: (S4_Cli; S9_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 19: (S4_Cli; S9_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 20: (S4_Cli; S9_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 21: (S4_Cli; S9_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 22: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 23: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 24: (S4_Cli; S9_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 25: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 26: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 27: (S4_Cli; S9_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 28: (S4_Cli; S10_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 29: (S4_Cli; S10_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 30: (S4_Cli; S10_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 31: (S4_Cli; S10_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 32: (S4_Cli; S10_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 33 (S4_Cli; S10_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 34 (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 35 (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 36: (S4_Cli; S10_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 37: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 38: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 39: (S4_Cli; S10_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 40: (S4_Cli; S10_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 41: (S4_Cli; S10_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 42: (S4_Cli; S10_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 43: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 44: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 45: (S4_Cli; S10_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 46: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 47: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 48: (S4_Cli; S10_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)

Cenário 49: (S4_Cli; S6_Cli) → (S3_M) → (S1_M2)
Cenário 50: (S5_Cli; S7_Cli) → (S3_M) → (S1_M2)
Cenário 51: (S5_Cli; S8_Cli) → (S3_M) → (S1_M2)
Cenário 52: (S5_Cli; S9_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 53: (S5_Cli; S9_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 54: (S5_Cli; S9_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 55: (S5_Cli; S9_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 56: (S5_Cli; S9_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)

Cenário 57: (S5_Cli; S9_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 58: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 59: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 60: (S5_Cli; S9_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 61: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 62: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 63: (S5_Cli; S9_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 64: (S5_Cli; S9_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 65: (S5_Cli; S9_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 66: (S5_Cli; S9_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 67: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 68: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 69: (S5_Cli; S9_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 70: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 71: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 72: (S5_Cli; S9_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 73: (S5_Cli; S10_Cli; S11_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 74: (S5_Cli; S10_Cli; S12_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 75: (S5_Cli; S10_Cli; S13_Cli) → (S1_M) → (S3_M2) → (S2_Serv)
Cenário 76: (S5_Cli; S10_Cli; S14_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 77: (S5_Cli; S10_Cli; S14_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 78 (S5_Cli; S10_Cli; S14_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 79 (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 80 (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 81: (S5_Cli; S10_Cli; S14_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 82: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 83: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 84: (S5_Cli; S10_Cli; S14_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 85: (S5_Cli; S10_Cli; S15_Cli; S16_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 86: (S5_Cli; S10_Cli; S15_Cli; S17_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 87: (S5_Cli; S10_Cli; S15_Cli; S18_Cli) → (S1_M) → (S3_M2; S2_M2) → (S1_Serv)
Cenário 88: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 89: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 90: (S5_Cli; S10_Cli; S15_Cli; S19_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 91: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S21_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 92: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S22_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)
Cenário 93: (S5_Cli; S10_Cli; S15_Cli; S20_Cli; S23_Cli) → (S1_M; S2_M) → (S3_M2; S4_M2) → (S1_Serv)

B.5 Cenários para a Escrita no Arquivo

Operações de Escrita no Arquivo bem sucedidas

S1_Cli: arquivo é escrito sem nenhum problema
(ABERTO)gfs_write[U](ESCREVENDO)gfs_swrite[S]→
(ESCREVENDO)confirmescrita_OK[S](ABERTO)return_OK[U]
S2_Cli: Cliente retransmite pedido de escrita até conseguir resposta do Servidor
(ABERTO) gfs_write [U](ESCREVENDO) gfs_swrite [S] → (ESCREVENDO)_(ESCREVENDO) gfs_swrite [S] → ... → (ESCREVENDO)_(ESCREVENDO) gfs_swrite [S] → (ESCREVENDO)confirmescrita_OK[S] (ABERTO)return_OK[U]

Situação nos Servidores

S1_Serv: Servidor lê arquivo com sucesso
(ABERTO) gfs_swrite [C](ABERTO) confirmescrita_OK [C]

Cenários que levam à leitura do arquivo com sucesso
Cenário1: (S1_Cli) → (S1_Serv)
Cenário2: (S2_Cli) → (S1_Serv)

Operações de Escrita no Arquivo mal sucedidas

S1_Cli: Servidor retorna ERRO ao Cliente na primeira transmissão
(ABERTO)gfs_write[U](ESCREVENDO)gfs_swrite[S] → (ESCREVENDO)confirmescrita_ERRO[S] (ABERTO)return_ERRO[U]
S2_Cli: Cliente retransmite pedido até contactar o Servidor, porém recebe ERRO como resposta
(ABERTO)gfs_write[U](ESCREVENDO)gfs_swrite[S] → (ESCREVENDO)_(ESCREVENDO) gfs_swrite [S] → ... →
(ESCREVENDO)_(ESCREVENDO)gfs_swrite[S] → (ESCREVENDO)confirmescrita_ERRO[S] (ABERTO)return_ERRO[U]
S3_Cli: Cliente retransmite pedido, mas não consegue contactar Mestre
(ABERTO)gfs_write[U](ESCREVENDO)gfs_swrite[S] → (ESCREVENDO)_(ESCREVENDO)gfs_swrite[S] → ... →
(ESCREVENDO)_(ESCREVENDO)gfs_swrite[S] →
(ESCREVENDO)_ (ABERTO)return_ERRO[U]

Situação nos Servidores

S1_Serv: Servidor não consegue escrever no arquivo
(ABERTO)gfs_swrite[C](ABERTO)confirmescrita_ERRO[C]

Cenários que levam à falha na escrita no arquivo
Cenário1: (S1_Cli) → (S1_Serv)
Cenário2: (S2_Cli) → (S1_Serv)
Cenário3: (S3_Cli)

B.6 Cenários para a Leitura do Arquivo

Operações de Leitura do Arquivo bem sucedidas

S1_Cli: arquivo é lido sem nenhum problema
(ABERTO)gfs_read[U](LENDO)gfs_sread[S] → (LENDO)confirmdado_OK[S] (ABERTO)return_OK[U]
S2_Cli: Cliente retransmite pedido de leitura até conseguir resposta do Servidor
(ABERTO)gfs_read[U](LENDO)gfs_sread[S] → (LENDO)_(LENDO)gfs_sread[S] → ... →
(LENDO)_(LENDO)gfs_sread[S] → (LENDO)confirmdado_OK[S] (ABERTO)return_OK[U]

Situação nos Servidores

S1_Serv: Servidor lê arquivo com sucesso
(ABERTO)gfs_sread[C](ABERTO)confirmdado_OK[C]

Cenários que levam à leitura do arquivo com sucesso
Cenário1: (S1_Cli) → (S1_Serv)
Cenário2: (S2_Cli) → (S1_Serv)

Operações de Leitura do Arquivo mal sucedidas

S1_Cli: Servidor retorna ERRO ao Cliente na primeira transmissão
(ABERTO)gfs_read[U](LENDO)gfs_sread[S] → (LENDO)confirmdado_ERRO[S] (ABERTO)return_ERRO[U]
S2_Cli: Cliente retransmite pedido até contactar o Servidor, porém recebe ERRO como resposta
(ABERTO)gfs_read[U](LENDO)gfs_sread[S] → (LENDO)_(LENDO)gfs_sread[S] → ... →
(LENDO)_(LENDO)gfs_sread[S] → (LENDO)confirmdado_ERRO[S] (ABERTO)return_ERRO[U]

S3_Cli: Cliente retransmite pedido, mas não consegue contactar Mestre
(ABERTO)gfs_read[U](LENDO)gfs_sread[S] → (LENDO)_(LENDO)gfs_sread[S] → ... →
(LENDO)_(LENDO)gfs_sread[S] →
(LENDO)_ (ABERTO)return_ERRO[U]

S1_Serv: Servidor não consegue ler arquivo
(ABERTO)gfs_read[C](ABERTO)confirmdado_ERRO[C]

Cenários que levam à falha na leitura de arquivo
Cenário1: (S1_Cli) → (S1_Serv)
Cenário2: (S2_Cli) → (S1_Serv)
Cenário3: (S3_Cli)

Posicionar Arquivo: operação bem e mal sucedida

S1_Cli:arquivo é posicionado corretamente
(ABERTO) gfs_lseek[U](ABERTO)return_OK

S2_Cli: ocorre um erro ao posicionar o arquivo
(ABERTO)gfs_lseek[U](ABERTO)return_ERRO

Cenários que ocorrem devido ao ajuste da posição do arquivo

Cenário1: (S1_Cli) { operação bem sucedida }
Cenário2: (S2_Cli) { operação mal sucedida }

Anexo C: Especificação Formal do Sistema Descentralizado

```
specification Galley;

default individual queue;
timescale seconds;

const
   N_usuarios = 10; { numero de usuarios do Sistema }
   Cli = 10; {numero maximo de arquivos que um cliente pode manipular simultaneamente}
   Serv = 10;{ numero maximo de arquivos que um servidor pode manipular
               simultaneamente}
   Mtr = 30; { numero maximo de arquivos que o Mestre pode manipular simultaneamente }
   N_servidores = 4;  { numero de usuarios do Sistema }
   N_arquivos = 50;   { numero maximo de arquivos que o Mestre pode gerenciar }
   MAXCOMPR = 255;    { comprimento maximo do PATH de um arquivo }
   IndiceInvalido = -1;         { valor retornado para indice invalido }
   Compr_lista_serv = 9;        { N_servidores+1 }
   Compr_lista_cli = 11;        { N_usuarios+1 }
   NRO_MAX_SEQ = 32000;         { limite maximo para a numeracao de mensagens }

   ALTA = 0;                    { usado como prioridade alta }
   MEDIA = 1;                   { usado como prioridade media }
   BAIXA = 2;                   { usado como prioridade baixa }

type
   serv_list = array[1..Compr_lista_serv] of integer;
   TipoDado = integer;
   TipoArquivo = array[1..MAXCOMPR] of char;
   TipoUnidade = integer;
   fd = integer;
   espera = boolean;
   TipoBuffer = integer;
   size_t = integer;
   Tipo_offset_count = integer;
   Tipo_str_unit = integer;
   tipomodulo = (mod_usuario, mod_cliente, mod_rede_cliente, mod_mestre,
                 mod_rede_mestre, mod_servidor, mod_rede_servidor);

   info_serv = record                      { usado pelos clientes }
                 ident_serv:integer;
                 confirmado:boolean;
               { dado enviado para cada servidor no caso de uma operacao de escrita }
                 dados:TipoDado;
               end;
   TipoListaServCli = array[1..Compr_lista_serv] of info_serv;

   info_serv_oper = record                  { usado pelo Mestre }
                      ident_serv:integer;
                      confirmado:boolean;
                    end;
   TipoListaServ = array[1..Compr_lista_serv] of info_serv_oper;

   TipoNumeracaoMsg = integer;  { tipo usado na numeracao de mensagens }
   info_cli = record
                 ident_inst, ident_cli:integer;
                 nro_seq:TipoNumeracaoMsg;
              end;
   cli_list = array[1..Compr_lista_cli] of info_cli;

   TipoStatus = (ARQ_ABERTO, ARQ_FECHADO, ARQ_ABRINDO, ARQ_FECHANDO, ARQ_REMOVENDO,
                 ARQ_REMOVIDO, ARQ_RENOMEANDO, ARQ_RENOMEADO, RENOMEADO_M1,
                 RENOMEADO_M2, OK, ERRO, PATH_INVALIDO, LISTA_DE_SERVIDORES_INVALIDA,
                 ERRO_FECHADO, ERRO_UN_REN_CL, FID_INVALIDO, ERRO_ARQABERTO,
                 ERRO_APAGANDO, ERRO_RENOMEANDO,    ERRO_ARQINEXISTENTE, ERRO_UNLINK,
                 ERRO_RENAME, ERRO2, NOMES_ARQ_IGUAIS,
                 ERRO_NOVO_NOME_JA_EXISTE_NO_SISTEMA, ERRO_OPERACAO_ILEGAL);
```

```
TipoOperacao = (ABERTO, ABRINDO, CRIANDO, RENOMEANDO, REMOVENDO, FECHANDO, FECHADO,
                  LENDO, ESCREVENDO, DESOCUPADO);
   tipo_var_sim = record    { usado pela simulacao }
                    r_open, r_close, r_unlink, r_rename: fd;
                    r_read, r_write:TipoStatus;
                  end;

{ declaracao dos canais de comunicacao }

channel UsuarioCliente(usuario,provedor);
by usuario: gfs_open(arq: tipoarquivo ; modo: integer ; tam_unid: Tipo_str_unit ; lista_s:
serv_list );
            gfs_unlink(arq:TipoArquivo; espera:boolean);
            gfs_close(fid:fd; espera:boolean);
            gfs_rename(arq_velho, arq_novo:TipoArquivo; espera:boolean);
            gfs_lseek(fid: fd ; offset: Tipo_offset_count ; origem: integer );
            gfs_read(fid: fd ; buf: tipobuffer ; count: Tipo_offset_count );
            gfs_write(fid: fd ; buf: tipobuffer ; count: Tipo_offset_count );

by provedor:confirm_gfs_open(fid:fd);
            confirm_gfs_unlink(arq:TipoArquivo; status:TipoStatus);
            confirm_gfs_rename(arq:TipoArquivo; status:TipoStatus; f_serv:boolean);
            respread(fid: integer ; buffer: tipobuffer ; n_bytes: Tipo_offset_count ;
                     status: tipostatus );
            respwrite(fid:fd; n_bytes:integer; status:TipoStatus);
            confirm_gfs_close(fid:fd);
            confirm_gfs_lseek(fid:fd; status:TipoStatus);

channel ClienteRede(usuario,provedor);
by usuario:gfs_sopen(fid:fd; arq:TipoArquivo; modo:integer; lista_serv:TipoListaServ;
                     ind_inst, ind_cli:integer; nro_seq:TipoNumeracaoMsg);
           gfs_sclose(fid:fd;  lista_serv:TipoListaServ; ind_inst, ind_cli:integer;
                      nro_seq:TipoNumeracaoMsg);
           gfs_sunlink(fid:fd; lista_serv:TipoListaServ; ind_inst, ind_cli:integer;
                       nro_seq:TipoNumeracaoMsg);
           gfs_srename(fid:fd; arq_velho, arq_novo:TipoArquivo;
                       lista_serv:TipoListaServ;  ind_inst, ind_cli:integer;
                       nro_seq:TipoNumeracaoMsg);
           gfs_sread(fid:fd; lista_serv:TipoListaServCli; ind_cli:integer;
                     nro_seq:TipoNumeracaoMsg);
           gfs_swrite(fid:fd; lista_serv:TipoListaServCli;ind_cli:integer;
                      nro_seq:TipoNumeracaoMsg);
           reg_file(arq:TipoArquivo; modo:integer; tam_unid:TipoUnidade;
                    lista_serv:serv_list; ind_inst, ind_cli, ind_serv:integer;
                    nro_seq:TipoNumeracaoMsg);
           reg_file_status(fid:fd; status:TipoStatus; ind_inst, ind_cli,
                           ind_serv:integer; nro_seq:TipoNumeracaoMsg);
           reg_file_status_name(arq:TipoArquivo; status:TipoStatus; ind_inst, ind_cli,
                                ind_serv:integer; nro_seq:TipoNumeracaoMsg);
           reg_file_status_ren(arq:TipoArquivo; lista_serv:serv_list;
                               tam_unid:TipoUnidade; ind_inst, ind_cli,
                               ind_serv:integer; nro_seq:TipoNumeracaoMsg);
           reg_file_status_rename(arq:TipoArquivo; fid:fd; status:TipoStatus;
                                  ind_inst, ind_cli, ind_serv:integer;
                                  nro_seq:TipoNumeracaoMsg);

by provedor:confirm_sopen(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                          ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            confirmdado(fid:fd; ind_serv:integer; dados:TipoDado; status:TipoStatus;
                        nro_seq:TipoNumeracaoMsg);
            confirmescrita(fid:fd; ind_serv:integer; status:TipoStatus;
                           ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            confirm_sclose(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                           ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            confirm(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                    ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            confirm_reg_file(fid:fd; tam_unid:TipoUnidade; servidores:serv_list;
```

```
                              status:TipoStatus; ind_inst, ind_cli:integer;
                              nro_seq:TipoNumeracaoMsg);
            confirm_reg_status(fid:fd; status:TipoStatus; ind_inst, ind_cli:integer;
                               lista_serv:serv_list; nro_seq:TipoNumeracaoMsg);
            confirm_reg_arq(fid:fd; status:TipoStatus; tam_unid:TipoUnidade;
                            lista_serv:serv_list; ind_inst, ind_cli:integer;
                            nro_seq:TipoNumeracaoMsg);
            confirm_reg_na(fid:fd; status:TipoStatus; ind_inst, ind_cli:integer;
                           nro_seq:TipoNumeracaoMsg);

channel ServidorRede(usuario,provedor);
by usuario: gfs_open(fid:fd; arq:TipoArquivo; modo:integer; ind_inst, ind_cli:integer;
                     nro_seq:TipoNumeracaoMsg);
            gfs_close(fid:fd; ind_inst, ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            gfs_unlink(fid:fd; arq:TipoArquivo; ind_inst, ind_cli:integer;
                       nro_seq:TipoNumeracaoMsg);
            gfs_rename(fid:fd; arq1, arq2:TipoArquivo; ind_inst, ind_cli:integer;
                       nro_seq:TipoNumeracaoMsg);
            gfs_read(fid:fd; ind_cli:integer; nro_seq:TipoNumeracaoMsg);
            gfs_write(fid:fd; dados:TipoDado; ind_cli:integer;
                      nro_seq:TipoNumeracaoMsg);
            reg_file(arq:TipoArquivo; modo:integer; tam_unid:TipoUnidade;
                     lista_serv:serv_list; ind_inst, ind_cli, ind_serv:integer;
                     nro_seq:TipoNumeracaoMsg);
            reg_file_status_serv(fid:fd; status:TipoStatus; ind_inst, ind_cli:integer;
                                 nro_seq:TipoNumeracaoMsg);
            reg_file_status_name(arq:TipoArquivo; status:TipoStatus; ind_inst,
                                 ind_cli, ind_serv:integer; nro_seq:TipoNumeracaoMsg);
            reg_file_status_rename(arq:TipoArquivo; fid:fd; status:TipoStatus;
                                   ind_inst, ind_cli, ind_serv:integer;
                                   nro_seq:TipoNumeracaoMsg);
            reg_file_status_ren(arq:TipoArquivo; lista_serv:serv_list;
                                tam_unid:TipoUnidade; ind_inst, ind_cli,
                                ind_serv:integer; nro_seq:TipoNumeracaoMsg);

by provedor: confirm_sopen(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                           ind_cli:integer; nro_seq:TipoNumeracaoMsg);
             confirm_sclose(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                            ind_cli:integer; nro_seq:TipoNumeracaoMsg);
             confirm(fid:fd; ind_serv:integer; status:TipoStatus; ind_inst,
                     ind_cli:integer; nro_seq:TipoNumeracaoMsg);
             confirm_dado(fid:fd; ind_serv, ind_cli:integer; dados:TipoDado;
                          status:TipoStatus; nro_seq:TipoNumeracaoMsg);
             confirm_escrita(fid:fd; ind_serv,ind_cli:integer; status:TipoStatus;
                             nro_seq:TipoNumeracaoMsg);
             confirm_reg_file_serv(fid:fd; tam_unid:TipoUnidade; servidores:serv_list;
                                   status: TipoStatus; lista_cli: cli_list);
             confirm_reg_status_serv(fid:fd; status:TipoStatus; lista_serv: serv_list;
                                     lista_cli: cli_list);
             confirm_reg_arq(fid:fd; status:TipoStatus; tam_unid:TipoUnidade;
                             lista_serv:serv_list; ind_inst, ind_cli:integer;
                             nro_seq:TipoNumeracaoMsg);
             confirm_reg_na(fid:fd; status:TipoStatus; ind_inst, ind_cli:integer;
                            nro_seq:TipoNumeracaoMsg);


{ definicao do usuario }

   module Usuario systemprocess (ind_cli: integer );
      ip u: usuariocliente(usuario) individual queue;
   end;

    body CorpoUsuario for Usuario;

        TYPE
          transicao = (open, close, read, write, rename, unlink, seek, none);

      var x: serv_list ;
```

```
var arq, arq_n, arq1, arq2, arq3, arq4: tipoarquivo ;
var fid, modo, fid_aux, indice, indice2: integer ;
var count: Tipo_offset_count;
var buf: tipobuffer ;
var esp: boolean ;
var opcao: integer ;
var ult_oper_c: transicao ;
var tempo_execucao: real ;
var a, offset, origem: integer ;
var tam: Tipo_str_unit;
var ativar, desativar, macro_simulacao, entraparametros, p_buffer,
    receber_confirm: boolean ;
var este_modulo: tipomodulo ;
var n_bytes_or, count_r, count_req:Tipo_offset_count;

state
  inativo , ativo, lendo, escrevendo ;

procedure preenchevetor(var vet: tipoarquivo );

  var i: integer ;
  begin
    for i := 11 to maxcompr DO
      vet [i ] := ' ';
  END ;

{ procedure preenchebuffer(var buf: tipobuffer );

  var i: integer ;
  begin
    i := 1 ;
    repeat begin
      buf [i ] := 10;
      buf [i + 1 ] := 0;
      i := i + 2 ;
    end  until ( i > maxbuf ) ;
  end ; }

function menu: integer;
primitive;

procedure obtemarq(a: integer ; var buffer: tipobuffer );
primitive;

procedure exibemensagem(cod: integer ; fid: integer ; status: tipostatus ;
                        ind_cli: integer );
primitive;

procedure exibebuffer(n_bytes: integer ; buffer: tipobuffer );
primitive;

procedure entraparametrosabrir(var arq: tipoarquivo ; var modo: integer ;
                               var tam: Tipo_str_unit );
primitive;

procedure entraparametrosfechar(var fid: integer );
primitive;

procedure entraparametrosler(var fid: integer;
                             var count_req: Tipo_offset_count);
primitive;

procedure entraparametrosescrever(var fid: integer; var count_req:
                                  Tipo_offset_count);
primitive;

procedure entraparametrosremover(var arq: tipoarquivo );
primitive;
```

```
procedure entraparametrosrenomear(var arq1: tipoarquivo ;
                                  var arq2: tipoarquivo );
primitive;

procedure entraparametrosajustarponteiro(var fid: integer ;
                    var offset: integer ; var origem: integer );
primitive;

procedure exibenrobytes(n_bytes: Tipo_offset_count );
primitive;

procedure ajustaserv(var serv: serv_list );
primitive;

procedure entra_nome_arquivo(var nome_arquivo: tipoarquivo; var fid:integer);
primitive;

procedure le_origem(var n_bytes: Tipo_offset_count; var buffer:tipobuffer;
                   fid:integer; count_req:Tipo_offset_count);
primitive;

procedure escreve_origem(buffer:tipobuffer; fid:integer;
                         n_bytes_w:Tipo_offset_count);
primitive;

procedure entra_nome_arquivo_destino(var fid:integer);
primitive;

procedure iniciatempo;
primitive;

procedure imprimetempo;
primitive;

initialize to inativo

  begin
    ult_oper_c := none ;
    x [1 ] := 1 ;
    x [2 ] := 2 ;
    x [3 ] := 3 ;
    x [4 ] := 4 ;
    x [5 ] := -1 ;
    x [6 ] := 6 ;
    x [7 ] := 7 ;
    x [8 ] := 8 ;
    x [9 ] := -1 ;
    arq1 [1 ] := 'a';
    arq1 [2 ] := ' ';
    preenchevetor ( arq1 ) ;
    arq2 [1 ] := 'k';
    arq2 [2 ] := ' ';
    preenchevetor ( arq2 ) ;
    arq3 [1 ] := 'a';
    arq3 [2 ] := 'b';
    arq3 [3 ] := 'c';
    arq3 [4 ] := 'd';
    arq3 [5 ] := 'e';
    arq3 [6 ] := 'f';
    arq3 [7 ] := 'g';
    arq3 [8 ] := 'h';
    arq3 [9 ] := 'i';
    arq3 [10 ] := 'j';
    arq4 [1 ] := 'k';
    arq4 [2 ] := 'l';
    arq4 [3 ] := 'm';
    arq4 [4 ] := 'n';
```

```
    arq4 [5 ] := 'o';
    arq4 [6 ] := 'p';
    arq4 [7 ] := 'q';
    arq4 [8 ] := 'r';
    arq4 [9 ] := 's';
    arq4 [10 ] := 't';
    fid := 0 ;
    count := 1000 ;   { valor default }
    indice := 1 ;
    indice2 := 1 ;
    esp := true ;
    tam := 120 ;
    tempo_execucao := 0.0 ;
    ativar := false ; { ------- false para simulacao ---------- }
    desativar := false ;
    macro_simulacao := false ;
    p_buffer := false ;
    opcao := 0 ;
    este_modulo := mod_usuario ;
    entraparametros := true ;
    offset := 0 ;
    origem := 1 ;
  end ;

trans
  from inativo to ativo
  provided ativar
  begin
  end ;

trans
  from ativo to inativo
  PRIORITY alta
  provided desativar
  begin
    ativar := false ;
    desativar := false ;
  end ;

trans
  from ativo to same
  provided ( opcao = 0 )
  begin
    if not ( macro_simulacao ) then
      opcao := menu
    else
      {macro_simulacao := false ;}
  end ;

trans
  from ativo to SAME
  provided ( opcao = 1 )
  begin
    if entraparametros then
      entraparametrosabrir ( arq , modo , tam )
    else
      arq [1 ] := arq3 [indice ] ;
    output u.gfs_open(arq , modo , tam , x ) ;
    opcao := -1 ;
  end ;

trans
  from ativo to SAME
  when u.confirm_gfs_open(fid)
  provided (fid >= 0)
  begin
    fid_aux := fid ;
    exibemensagem ( 1, fid_aux, OK, ind_cli ) ;
```

```
    ult_oper_c := open ;
    if not ( macro_simulacao ) then
      opcao := 0 ;
  end ;

 trans
  from ativo to SAME
  when u.confirm_gfs_open(fid)
  provided ( fid < 0 )
  begin
    exibemensagem( 2 ,fid ,OK ,ind_cli) ;
    ult_oper_c := open ;
    if not ( macro_simulacao ) then
      opcao := 0 ;
  end ;

 trans
  from ativo to escrevendo
  provided ( opcao = 4 )
  begin
    if entraparametros then
      begin
        entraparametrosescrever ( fid, count_req ) ;
        {entra_nome_arquivo(arq, fid_aux);}
        {le_origem(n_bytes_or, buf, fid_aux, count_req);}
      end
    else
      if p_buffer then
         begin
           entra_nome_arquivo(arq, fid_aux);
           le_origem(n_bytes_or, buf, fid_aux, count_req);
         end;
      {iniciatempo;}   { inicia contagem de tempo para o teste }
      output u.gfs_write(fid ,buf ,count_req);
    receber_confirm := true;
  end ;


 trans
  from escrevendo to ativo
  when u .respwrite ( fid, n_bytes, status )
  {provided (status <> OK)}  { Erro ao escrever no arquivo }
  begin
    {imprimetempo;}
    ult_oper_c := write ;
    exibemensagem(4, fid, status, 1);
    exibenrobytes ( n_bytes_or ) ;
    opcao := -1 ;
  end ;

trans
  from ativo to lendo
  provided ( opcao = 3 )
  begin
    if entraparametros then
    begin
      entraparametrosler ( fid, count_req ) ;

    end;
    {iniciatempo;}
    output u.gfs_read (fid ,buf ,count_req)
    {if count_r > count_req then
       output u.gfs_read (fid ,buf ,count_req)
    else
       output u.gfs_read (fid ,buf ,count_r );}
    {opcao := -1 ;}
  end ;
```

```
trans
  from lendo to ativo
  when u.respread ( fid, buffer, n_bytes, status )
  {provided n_bytes <= 0 }
  begin
    ult_oper_c := read ;
    exibemensagem(3, fid, status, 1);
    exibenrobytes ( n_bytes) ;
    opcao := -1 ;
  end ;


trans
  from ativo
  to same
  provided ( opcao = 2 )
  begin
    if entraparametros then
      entraparametrosfechar ( fid ) ;
    output u.gfs_close ( fid , esp ) ;
    opcao := -1 ;
  end ;

trans
  from ativo to same
  when u.confirm_gfs_close(fid)
  begin
    ult_oper_c := close ;
    exibemensagem ( 5 , fid , OK , ind_cli ) ;
    if not ( macro_simulacao ) then
      opcao := 0 ;
  end ;

trans
  from ativo
  to same
  provided ( opcao = 7 )
  begin
    if entraparametros then
      entraparametrosajustarponteiro ( fid , offset , origem ) ;
    output u.gfs_lseek ( fid , offset , origem ) ;
    opcao := -1 ;
  end ;

trans
  from ativo
  to same
  when u.confirm_gfs_lseek ( fid, status )
  begin
    ult_oper_c := seek ;
    exibemensagem ( 6 , -1 , status , ind_cli ) ;
    if not ( macro_simulacao ) then
      opcao := 0 ;
  end ;

trans
  from ativo to same
  provided ( opcao = 5 )
  begin
    if entraparametros then
      entraparametrosrenomear ( arq , arq_n )
    else
      begin
        arq [1 ] := arq3 [indice ] ;
        arq_n [1 ] := arq4 [indice2 ] ;
      end ;
    output u.gfs_rename ( arq , arq_n , esp ) ;
```

```
        opcao := -1 ;
      end ;

    trans
      from ativo to same
      when u.confirm_gfs_rename ( arq, status, f_serv )
      begin
        ult_oper_c := rename ;
        exibemensagem ( 7 , -1 , status , ind_cli ) ;
        if not ( macro_simulacao ) then
          opcao := 0 ;
      end ;

    trans
      from ativo to same
      provided ( opcao = 6 )
      begin
        if entraparametros then
          entraparametrosremover ( arq )
        else
          arq [1 ] := arq3 [indice ] ;
        output u.gfs_unlink ( arq , esp ) ;
        opcao := -1 ;
      end ;

    trans
      from ativo to same
      when u.confirm_gfs_unlink ( arq, status )
      begin
        ult_oper_c := unlink ;
        exibemensagem ( 8 , -1 , status , ind_cli ) ;
        if not ( macro_simulacao ) then
          opcao := 0 ;
      end ;

    trans
     from ativo to same
      provided ( opcao = 8 )
      begin
        ajustaserv ( x ) ;
        if not ( macro_simulacao ) then
          opcao := 0 ;
      end ;

  end;  { of corpousuario }


{ definicao do Modulo Gerente }

module Gerente systemprocess;
   ip UC: array[1..N_usuarios] of UsuarioCliente(provedor);
end;

body CorpoGerente for Gerente;

{ definicao do Cliente ____________________________________________________ }

   module Cliente process (ind_cli:integer);
      ip C: UsuarioCliente(provedor);
         R: ClienteRede(usuario);
   end;

   body CorpoCliente for Cliente;
      ip  MA: array[1..Cli] of UsuarioCliente(usuario); { pontos de interacao
                                                          internos }
          MB: array[1..Cli] of ClienteRede(provedor);

      { definicao do modulo InstCli -> modulo(s) filho(s) de Cliente }
```

```
module InstCli process (ind_inst, ind_cli:integer);
   ip A: UsuarioCliente(provedor);
      B: ClienteRede(usuario);
end;

body CorpoInstCli for InstCli;
   const MAX_RETRANS = 10;      { nro maimo de retransmissoes }
         T = 200000;            { valor maximo de temporizacao }
   type
      info_msg = record
                 arq, arq2:TipoArquivo;
                 list_serv:serv_list;
                 f, f2:fd;
                 buf:TipoBuffer;
                 count: size_t;
                 espera:boolean;
                 status: TipoStatus;
                 modo:integer;
                 tam:TipoUnidade;
                 end;

   operacao = (REG_FILE, OPENING_FILE, REG_STATUS, CLOSING_FILE,
               REG_FILE_CLOSED, REG_CLOSING_FILE, REG_UNLINKING, UNLINKING,
               REG_UNLINKING_F, REG_RENAMING, CONTACTING_SEC_MS,
               CONFIRM_META_1, CONFIRM_META_2, UN_REG_RENAMING,
               UN_REG_RENAMING_2, RENAMING);

   var N_SERVERS, N_CONFIRM, N_BYTES, N_BLOCK, N_RETRANS: integer;
       RESP: boolean;
       { armazena servidores envolvidos com o arquivo aberto }
       lista_s: TipoListaServCli;
       lista_aux:serv_list;
       { lista dos servidores envolvidos com um determinado arquivo }
       serv_envolvidos:TipoListaServ;
       { variavel para armazenar mensagem a ser retransmitida }
       msg:info_msg;
       erro_oper:boolean;
       oper: operacao;
       { lista de servidores envolvidos com o arquivo }
       lista_servidores:serv_list;

   state ESPERA, REMOVENDO, RENOMEANDO, FECHANDO, ABRINDO, ABERTO, LENDO,
         ESCREVENDO;
   stateset IgnoraConfirmOpen = [ABERTO, LENDO, ESCREVENDO, FECHANDO,
                                 RENOMEANDO, REMOVENDO];
            IgnoraConfirmClose = [ABRINDO, ABERTO, LENDO, ESCREVENDO,
                                  RENOMEANDO, REMOVENDO];
            IgnoraConfirmUnlink = [ABRINDO, ABERTO, LENDO, ESCREVENDO,
                                   RENOMEANDO];
            IgnoraConfirmRename = [ABRINDO, ABERTO, LENDO, ESCREVENDO,
                                   REMOVENDO];
            IgnoraConfirmRead = [ABERTO, ESCREVENDO, ABRINDO, FECHANDO,
                                 RENOMEANDO, REMOVENDO];
            IgnoraConfirmWrite = [ABERTO, LENDO, FECHANDO, ABRINDO,
                                  RENOMEANDO, REMOVENDO];

   { definicao das funcoes }

   procedure ArmazenaServEnvolvidos(lista: serv_list);
   { armazena a lista de servidores envolvidos com o arquivo atualmente
     aberto }
   var i:integer;
   begin
      i := 1;
      while lista[i] <> -1 do
      begin
         lista_s[i].ident_serv := lista[i];
         lista_s[i].confirmado := false;
```

```
         i := i+1;
      end;
      lista_s[i].ident_serv := -1  { fim da lista }
   end; { ArmazenaServ }

   procedure RestServEnvolvidos;
   { restaura servidores para a condicao de nao-confirmados }
   var i:integer;
   begin
      i := 1;
      while lista_s[i].ident_serv <> -1 do
      begin
         lista_s[i].confirmado := false;
         i := i+1;
      end;
   end; { RestServEnvolvidos }

   function NroServEnvolvidos:integer;
   { determina o numero de servidores envolvidos com um determinado arquivo }
   var i:integer;
   begin
      i := 1;
      while serv_envolvidos[i].ident_serv <> -1 do
         i := i+1;
      NroServEnvolvidos := i-1
  end; { NroServEnvolvidos }

   procedure CopiaVet(lista:serv_list);
   { copia a lista de servidores a ser retransmitida durante as operacoes }
   var i:integer;
   begin
      i := 1;
      while lista[i] <> -1 do
      begin
         msg.list_serv[i] := lista[i];
         i := i+1;
      end;
      msg.list_serv[i] := -1
   end; { CopiaVet }

   procedure RecuperaListaServ(var list_aux:serv_list);
   { recupera lista de servidores para fazer retransmissao }
   var i:integer;
   begin
      i := 1;
      while msg.list_serv[i] <> -1 do begin
         list_aux[i] := msg.list_serv[i];
         i := i+1;
      end;
      list_aux[i] := -1
    end; { RecuperaListaServ }

   function VerificaServConfirm(ind_serv:integer):boolean;
   { verifica se um servidor ja' havia sido confirmado }
   var i:integer;
   begin
      i := 1;
      while lista_s[i].ident_serv <> ind_serv do
         i := i+1;
      if not lista_s[i].confirmado then
        { servidor ainda nao havia sido confirmado }
         VerificaServConfirm := false
      else
         VerificaServConfirm := true  { servidor ja' havia sido confirmado }
   end; { VerificaServConfirm }

   procedure AlteraServConfirm(ind_serv:integer; var l_s:TipoListaServ);
   { verifica se um servidor ja' havia sido confirmado }
```

```
var i:integer;
begin
   i := 1;
   while l_s[i].ident_serv <> ind_serv do
      i := i+1;
   l_s[i].confirmado := true;      { atualiza confirmacao do servidor }
end; { AlteraServConfirm }

procedure AlteraServConfirmES(ind_serv:integer; l_s:TipoListaServCli);
{ verifica se um servidor envolvido numa operacao de E/S confirmou
  solicitacao de servico }
var i:integer;
begin
   i := 1;
   while l_s[i].ident_serv <> ind_serv do
      i := i+1;
   l_s[i].confirmado := true;      { atualiza confirmacao do servidor }
end; { AlteraServConfirmES }

procedure InverteConfServ(var serv_env:TipoListaServ);
{ inverte as confirmacoes dos servidores }
var i:integer;
begin
   i := 1;
   while serv_env[i].ident_serv <> -1 do
   begin
      serv_env[i].confirmado := false;
      i := i+1;
  end
end; { InverteConfServ }

function DadosServ(ind_serv:integer):TipoDado;
{ retorna os dados enviados para a escrita de um determinado servidor }
var i:integer;
begin
   i := 1;
   while lista_s[i].ident_serv <> ind_serv do
      i := i+1;
   DadosServ := lista_s[i].dados;
end; { DadosServ }

function ServidorEnvolvido(ident_serv:integer; buf:TipoBuffer;
                           count:size_t):boolean;
{ determina se um determinado servidor esta' envolvido com a leitura/escrita
  de um determinado arquivo. Se  operacao for de escrita (buf <> NIL), tambem
  distribui os dados para os respectivos servidores }
begin
   { distribuicao de dados deve ser feita durante a simulacao no Edb para a
     simulacao de diversos cenarios diferentes. Essa distribuicao e' feita
     diretamente no vetor de servidores quando a simulacao alcanca o estado
     adequado. }
   { valor de retorno default -> todos os servidores estarao envolvidos }
   ServidorEnvolvido := true;
end;

procedure DetUnidadesEnvolvidas(buf:TipoBuffer; count:size_t;
                           var l_s:TipoListaServCli; var n_blocos:integer);
{ determina os servidores envolvidos numa operacao de leitura/escrita }
var i:integer;
begin
   i := 1;
   { numero de servidores envolvidos com uma operacao de leitura/escrita }
   n_blocos := 0;
   while l_s[i].ident_serv <> -1 do
   begin
      if not ServidorEnvolvido(l_s[i].ident_serv, buf, count) then
         l_s[i].confirmado := true
      else begin
```

```
            n_blocos := n_blocos+1;
            l_s[i].confirmado := false;  { essa linha pode ser redundante }
      end;
      i := i+1;
   end
end; { DetUnidadesEnvolvidas }

function cmpstr(str1, str2:TipoArquivo):boolean;
{ verifica se duas strings sao iguais }
var i:integer;
begin
   i := 1;
   while (i <= MAXCOMPR) and (str1[i] = str2[i]) do
   i := i+1;
   if i > MAXCOMPR then
      cmpstr := true
   else
      cmpstr := false
end; { cmpstr }

procedure IniciaServEnvolvidos(lista_s:serv_list;
                               var serv_env:TipoListaServ);
{ Inicia a lista de servidores envolvidos }
var i:integer;
begin
   i := 1;
   while lista_s[i] <> -1 do begin
      serv_env[i].ident_serv := lista_s[i];
      serv_env[i].confirmado := false;
      i := i+1;
   end;
   serv_env[i].ident_serv := -1      { fim do vetor de servidores }
 end; { IniciaServEnvolvidos }

procedure IniciaListaServEnvolvidos(lista_serv:serv_list;
                                    var lista_serv_envolvidos: serv_list);
{ inicia lista simples de servidores }
var i:integer;
begin
   i := 1;
   while lista_serv[i] <> -1 do begin
      lista_serv_envolvidos[i] := lista_serv[i];
      i := i+1;
   end;
   lista_serv_envolvidos[i]:= -1      { fim do vetor de servidores }
 end; { IniciaListaServEnvolvidos }

procedure CopiaServEnvolvidos;
{ copia servidores envolvidos para lista_servidores }
var i:integer;
begin
   i := 1;
   while serv_envolvidos[i].ident_serv <> -1 do begin
      lista_servidores[i] := serv_envolvidos[i].ident_serv;
      i := i+1;
   end;
   lista_servidores[i] := -1   { fim da lista de servidores }
end; { CopiaServEnvolvidos }


function AvaliaNroBytes(dados:TipoDado):integer;
{ Retorna o numero de bytes de um bloco }
begin
   AvaliaNroBytes := 10 { valor default para a simulacao }
end;

procedure CopiaDadosBuffer(var buf:TipoBuffer; dados:TipoDado);
{ Copia os dados recebidos em um buffer }
```

```
begin
   { procedimento que nao gera nenhum resultado que interfira no
     funcionamento do protocolo }
end;

function AjustarPosicaoArquivo(fid:fd; lista_s:TipoListaServCli):TipoStatus;
{ implementa a funcao p_lseek }
begin
   AjustarPosicaoArquivo := OK  { valor de retorno default para a simulacao }
end;

function ComparaMsg(fid:fd; buf:TipoBuffer; count:size_t;
                    msg_original:info_msg):boolean;
{ Compara duas primitivas de E/S }
begin
   ComparaMsg := (fid = msg_original.f) and (buf = msg_original.buf) and
                 (count = msg_original.count)
end;

function Hash(arq: TipoArquivo):integer;
{ Dado o nome de um arquivo, a funcao retorna o numero do indice do servidor
  onde estao os metadados daquele arquivo }
primitive;


{ inicializacao }

initialize to ESPERA
begin
end;

{ inicio das transicoes }

trans
{ fase de abertura do arquivo }

 { 1. Entra em contato com metaserv para registrar arquivo e/ou obter
      metadados }

   from ESPERA to ABRINDO  { usuario solicita abertura do arquivo }
   when A.gfs_open(arq, modo, tam_unid, lista_s)
      begin
        { armazena informacoes da mensagem para fazer retransmissoes }
         msg.arq := arq;
         msg.modo := modo;
         msg.tam := tam_unid;
         CopiaVet(lista_s);
         N_RETRANS := 0;
         oper := REG_FILE;
         { contacta metaservidor para obter informacoes do arquivo a ser
           aberto/criado }
         output B.reg_file(arq, modo, tam_unid, lista_s, ind_inst, ind_cli,
                           hash(arq), 0) { 0 = nro_seq }
      end;

    from ABRINDO to same
    provided (N_RETRANS < MAX_RETRANS) and (oper = REG_FILE)
    priority BAIXA
    delay(T)
    begin
       N_RETRANS := N_RETRANS+1;
       RecuperaListaServ(lista_aux);
       output B.reg_file(msg.arq, msg.modo, msg.tam, lista_aux, ind_inst,
                    ind_cli, hash(msg.arq), 1)  { 1 -> retransmite mensagem }
    end;

   from ABRINDO to ESPERA
   provided (N_RETRANS =  MAX_RETRANS) and (oper = REG_FILE)
```

```
   priority BAIXA
   delay(T)
   begin
      output A.confirm_gfs_open(-1)  { erro na abertura do arquivo }
   end;

   from ABRINDO to same
   when B.confirm_reg_file(fid, tam_unid, servidores, status, ind_inst,
                           ind_cli, nro_seq)
   {verifica se houve problema durante o registro do arquivo no metaservidor}
   provided (fid >= 0) and (oper = REG_FILE) and (status = ARQ_ABRINDO)
   priority ALTA
      begin
         msg.f := fid;
         ArmazenaServEnvolvidos(servidores);
         { armazena os servidores envolvidos }
         IniciaServEnvolvidos(servidores, serv_envolvidos);
         N_SERVERS := NroServEnvolvidos;
         N_CONFIRM := 0;
         N_RETRANS := 0;
         oper := OPENING_FILE;
         { envia mensagem p_open a todos os servidores envolvidos }
         output B.gfs_sopen(msg.f, msg.arq, msg.modo, serv_envolvidos,
                            ind_inst, ind_cli, 0);
      end;


   from ABRINDO to ESPERA
   when B.confirm_reg_file(fid, tam_unid, servidores, status, ind_inst,
                           ind_cli, nro_seq)
   {verifica se houve problema durante o registro do arquivo no metaservidor}
   provided (fid < 0) and (oper = REG_FILE) and (status = ERRO)
   priority ALTA
      begin
         output A.confirm_gfs_open(-4)
      end;

  from ABRINDO to ABERTO
  when B.confirm_reg_file(fid, tam_unid, servidores, status, ind_inst,
                          ind_cli, nro_seq)
  {verifica se houve problema durante o registro do arquivo no metaservidor}
  provided (fid >= 0) and (oper = REG_FILE) and (status = ARQ_ABERTO)
  priority ALTA
     begin
        msg.f := fid;
        ArmazenaServEnvolvidos(servidores);
        { armazena os servidores envolvidos }
        IniciaServEnvolvidos(servidores, serv_envolvidos);
        N_SERVERS := NroServEnvolvidos;
       output A.confirm_gfs_open(fid);
     end;


{ 2. Entrando em contato com os servidores envolvidos para a abertura/criacao
     dos fragmentos locais do arquivo distribuido }


 from ABRINDO to same
 { todos os servidores confirmaram a abertura do arquivo }
 when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
 provided (status = OK) and (N_CONFIRM+1 = N_SERVERS) and
          (not VerificaServConfirm(ind_serv))
 { a ultima mensagem nao eh uma retransmissao }
 priority ALTA
    begin
       { atualiza confirmacao do servidor }
       AlteraServConfirm(ind_serv, serv_envolvidos);
       CopiaServEnvolvidos;
```

```
      { entra em contato com metaservidor para mudar o status do arquivo
        para aberto }
      oper := REG_STATUS;
      N_RETRANS := 0;
      output B.reg_file_status(fid, ARQ_ABERTO, ind_inst, ind_cli,
                           hash(msg.arq), 0); { 0 = nro_seq = primeira tx }
 end;

 from ABRINDO to same
 { todos os servidores confirmaram a abertura do arquivo }
 when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
 provided (status = OK) and (N_CONFIRM+1 = N_SERVERS) and
          (VerificaServConfirm(ind_serv))
 { a ultima mensagem e' uma retransmissao }
 priority ALTA
 begin
    { mensagem ignorada -> retransmissao }
 end;

 from ABRINDO to same
 provided (N_RETRANS < MAX_RETRANS) and (oper = REG_STATUS)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    { retransmite mensagem de registro de status }
    output B.reg_file_status(msg.f, ARQ_ABERTO, ind_inst, ind_cli,
                             hash(msg.arq), 1);
 end;


from ABRINDO to FECHANDO
provided (N_RETRANS =  MAX_RETRANS) and (oper = REG_STATUS)
priority BAIXA
delay(T)
begin
   output A.confirm_gfs_open(-1);  { erro ao abrir o arquivo )
   N_ACESSOS := 0;
   N_SERVERS := NroServEnvolvidos;
   { Restaura confirmacoes em serv_envolvidos }
   InverteConfServ(serv_envolvidos);
   { solicita aos servidores que haviam aberto o arquivo fechar seus
     respectivos segmentos }
   output B.gfs_sclose(msg.f, serv_envolvidos, ind_inst, ind_cli, 1);
   N_CONFIRM := 0;
   RESP := false;   { os Clientes ja' foram avisados }
   erro_oper := false; { usado para fechar o arquivo }
   { identifica fechamento de arquivo devido `a falha de abertura }
   oper := CLOSING_FILE;
end;

from ABRINDO to ABERTO  { abertura de arquivo bem sucedida }
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv,
                          nro_seq)
provided (status = ARQ_ABERTO) and (oper = REG_STATUS)
priority ALTA
begin
   output A.confirm_gfs_open(msg.f);
end;

from ABRINDO to ABERTO  { abertura de arquivo bem sucedida }
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv,
                          nro_seq)
{ arquivo estava sendo aberto por outro Cliente }
provided (status = ARQ_ABERTO) and (oper = REG_FILE)
priority ALTA
begin
   msg.f := fid;
```

```
      ArmazenaServEnvolvidos(lista_serv);
      { armazena os servidores envolvidos }
      IniciaServEnvolvidos(lista_serv, serv_envolvidos);
      N_SERVERS := NroServEnvolvidos;
      output A.confirm_gfs_open(msg.f);
   end;

   from ABRINDO to FECHANDO
   { status da abertura de arquivo nao registrada pelo meta-servidor }
   when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv,
                             nro_seq)
   { status registrado corretamente }
   provided (status <> ARQ_ABERTO) and (oper = REG_STATUS)
   priority ALTA
   begin
      output A.confirm_gfs_open(-2);  { erro ao abrir o arquivo )
      N_ACESSOS := 0;
      N_SERVERS := NroServEnvolvidos;
      { Restaura confirmacoes em serv_envolvidos }
      InverteConfServ(serv_envolvidos);
      { solicita aos servidores que haviam aberto o arquivo fechar seus
        respectivos segmentos }
      output B.gfs_sclose(fid, serv_envolvidos, ind_inst, ind_cli, 0);
      N_CONFIRM := 0;
      erro_oper := false; { usado para fechar o arquivo }
      { identifica fechamento de arquivo devido `a falha de abertura }
      oper := CLOSING_FILE;
      RESP := false;
   end;

  { Contatando Servidores ------------------------------------------------ }


from ABRINDO to same
when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
{ um dos servidores confirmou a abertura do arquivo }
provided (status = OK) and (N_CONFIRM+1 < N_SERVERS)
priority ALTA
begin
   if not VerificaServConfirm(ind_serv) then
   { atualiza confirmacao do servidor desde que a mensagem nao seja uma
     retransmissao }
   begin
      AlteraServConfirm(ind_serv, serv_envolvidos);
     { atualiza confirmacao do servidor }
      N_CONFIRM := N_CONFIRM+1;
   end
end;

from ABRINDO to FECHANDO
{ um dos servidores retornou ERRO ao tentar abrir o arquivo; mais de um
  servidor esta' envolvido com o arquivo }
when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = ERRO) and (N_SERVERS > 1)
priority ALTA
begin
   output A.confirm_gfs_open(-3);  { retorna ERRO ao Cliente }
   { Restaura confirmacoes em serv_envolvidos }
   InverteConfServ(serv_envolvidos);
   { servidor atual ja' respondeu }
   AlteraServConfirm(ind_serv, serv_envolvidos);
   N_SERVERS := N_SERVERS-1;
  { solicita aos servidores que haviam aberto o arquivo fechar seus
    respectivos segmentos }
   output B.gfs_sclose(msg.f, serv_envolvidos, ind_inst, ind_cli, 0);
   N_CONFIRM := 0;
   erro_oper := false; { usado para fechar o arquivo }
   oper := CLOSING_FILE;
```

```
    RESP := false;
 end;

from ABRINDO to same
{ o unico servidor envolvido responde ERRO ao pedido de abertura de arquivo }
when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = ERRO) and (N_SERVERS = 1)
priority ALTA
begin
    oper := REG_STATUS;
    { notifica meta-servidor que o arquivo deve ter o status alterado de
      abrindo para fechado }
    output B.reg_file_status(msg.f, ARQ_FECHADO, ind_inst, ind_cli,
                             hash(msg.arq), 0);
end;


from ABRINDO to same
provided (N_RETRANS < MAX_RETRANS) and (oper = OPENING_FILE)
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   { retransmite p_open para os servidores que ainda nao responderam }
   output B.gfs_sopen(msg.f, msg.arq, msg.modo, serv_envolvidos, ind_inst,
                      ind_cli, 1)
end;

from ABRINDO to FECHANDO
provided (N_RETRANS = MAX_RETRANS) and (N_SERVERS > 1) and
         (oper = OPENING_FILE)
priority BAIXA
delay(T)
begin
   output A.confirm_gfs_open(-1);   { retorna ERRO a o Cliente }
   { Restaura confirmacoes em serv_envolvidos }
   InverteConfServ(serv_envolvidos);
   { solicita aos servidores que haviam aberto o arquivo fechar seus
     respectivos segmentos }
   output B.gfs_sclose(msg.f, serv_envolvidos, ind_inst, ind_cli, 1);
   N_CONFIRM := 0;
   N_RETRANS := 0;
   erro_oper := false;
   oper := CLOSING_FILE;
   RESP := false;
end;

from ABRINDO to same
{ o unico servidor a responder nao respondeu em nenhuma retransmissao }
provided (N_RETRANS = MAX_RETRANS) and (N_SERVERS = 1) and
         (oper = OPENING_FILE)
priority BAIXA
delay(T)
begin
   oper := CLOSING_FILE;
   RESP := false;
   { altera status do arquivo no meta-servidor }
   output B.reg_file_status(msg.f, ARQ_FECHADO, ind_inst, ind_cli,
                            hash(msg.arq), 0);
end;



{ 3. Fechando arquivo ------------------------------------------------- }

from FECHANDO to same
when A.gfs_close(fid, espera)
priority ALTA
```

```
begin  { rx de pedido -> ignorar }
end;

from FECHANDO to same
when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
priority ALTA
begin  { mensagem fora de contexto -> ignorar }
end;

from FECHANDO to same
when A.gfs_open(arq, modo, tam_unid, lista_s)
priority ALTA
begin
   { impede pedido de abertura do arquivo atual durante operacao de fechamento
     de arquivo }
   output A.confirm_gfs_open(-1)
end;

from FECHANDO to same
{ um dos servidores confirma pedido para fechar arquivo }
when B.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = OK) and (N_CONFIRM+1 < N_SERVERS)
priority ALTA
begin
   { insere servidor na lista de servidores que responderam `a solicitacao }
   if not VerificaServConfirm(ind_serv) then
   begin
      { atualiza confirmacao do servidor }
      AlteraServConfirm(ind_serv, serv_envolvidos);
      N_CONFIRM := N_CONFIRM + 1;         { atualiza numero de confirmacoes }
   end
end;

from FECHANDO to same
{ todos os servidores confirmaram ao pedido para fechar arquivo }
when B.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
{ a ultima mensagem nao e' uma retransmissao }
provided (N_CONFIRM+1 = N_SERVERS)  and (not VerificaServConfirm(ind_serv))
priority ALTA
{ Cliente ja' recebeu resposta de erro de abertura. Operacao de fechamento nao
  responde ao Cliente }
begin
   { atualiza confirmacao do servidor }
   AlteraServConfirm(ind_serv, serv_envolvidos);
   N_RETRANS := 0;
   oper := REG_FILE_CLOSED;
   if status <> OK then
      erro_oper := true;  { sinaliza erro no fechamento do arquivo }
   { informa o meta-servidor que arquivo foi fechado corretamente }
   output B.reg_file_status(msg.f, ARQ_FECHADO, ind_inst, ind_cli,
                            hash(msg.arq), 0);
end;

from FECHANDO to same
{ todos os servidores confirmaram ao pedido para fechar arquivo }
when B.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
{ a ultima mensagem e' uma retransmissao }
provided (N_CONFIRM+1 = N_SERVERS) and (VerificaServConfirm(ind_serv))
priority ALTA
begin
   { ignora mensagem }
end;

from FECHANDO to same
{ um dos servidores retorna ERRO ao pedido para fechar arquivo }
when B.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = ERRO) and (N_CONFIRM+1 < N_SERVERS)
priority ALTA
```

```
begin
   if not VerificaServConfirm(ind_serv) then
   begin
      { atualiza confirmacao do servidor }
      AlteraServConfirm(ind_serv, serv_envolvidos);
      N_CONFIRM := N_CONFIRM+1;  { atualiza numero de confirmacoes }
      erro_oper := true;  { sinaliza ero no fechamento do arquivo }
  end
end;

from FECHANDO to same
provided (N_RETRANS < MAX_RETRANS) and (oper = CLOSING_FILE)
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   { retransmite p_close para os servidores que ainda nao responderam }
   output B.gfs_sclose(msg.f, serv_envolvidos, ind_inst, ind_cli, 1)
end;

from FECHANDO to same
provided (N_RETRANS = MAX_RETRANS) and (oper = CLOSING_FILE)
priority BAIXA
delay(T)
begin
   N_RETRANS := 0;
   erro_oper := true;  { sinaliza que nem todos os servidores responderam }
   { De qualquer modo, registra operacao no meta-serv }
   oper := REG_FILE_CLOSED;
   {informa o meta-servidor que a operacao terminou }
   output B.reg_file_status(msg.f, ARQ_FECHADO, ind_inst,
                            ind_cli,hash(msg.arq), 0);
end;

{ retornar ao estado inicial somente apos confirmacao do meta-servidor }

from FECHANDO to ESPERA
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv, nro_seq)
provided (oper = REG_FILE_CLOSED){ operacao confirmada pelo meta-servidor }
priority ALTA
 begin
    if (RESP) then  begin { somente responder se a operacao for sincrona }
       if status <> ARQ_FECHADO then
          output A.confirm_gfs_close(-5)  { erro ao registrar operacao }
       else
          if erro_oper then
             output A.confirm_gfs_close(-5)
          else
             output A.confirm_gfs_close(0)  { operacao teve exito }
    end;
 end;

from FECHANDO to same
provided (N_RETRANS < MAX_RETRANS) and (oper = REG_FILE_CLOSED)
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   output B.reg_file_status(msg.f, ARQ_FECHADO, ind_inst, ind_cli,
                            hash(msg.arq), 1); { rx mensagem ao meta-servidor }
end;

from FECHANDO to ESPERA
provided (N_RETRANS = MAX_RETRANS) and (oper = REG_FILE_CLOSED)
priority BAIXA
delay(T)
begin
   if (RESP) then
```

```
      output A.confirm_gfs_close(-1);  { falha ao contactar o meta-servidor }
end;


{ controle da primeira notificacao ao meta-servidor quanto ao fechamento de
  arquivo }


from FECHANDO to same
provided (N_RETRANS < MAX_RETRANS) and (oper = REG_CLOSING_FILE)
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   output B.reg_file_status(msg.f, ARQ_FECHANDO, ind_inst, ind_cli,
                            hash(msg.arq), 1);
end;

from FECHANDO to ESPERA
provided (N_RETRANS = MAX_RETRANS) and (oper = REG_CLOSING_FILE)
priority BAIXA
delay(T)
begin
   if (RESP) then
      output A.confirm_gfs_close(-1);
end;

from FECHANDO to same
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv, nro_seq)
{ MetaServidor registrou inicio da operacao de fechamento de arquivo }
provided (oper = REG_CLOSING_FILE) and (status = ARQ_FECHANDO)
priority ALTA
begin
   { notificacao teve exito; entrar em contato com os servidores }
   { Restaura confirmacoes em serv_envolvidos }
   InverteConfServ(serv_envolvidos);
   { solicita aos servidores que haviam aberto o arquivo fechar seus
     respectivos segmentos }
    output B.gfs_sclose(msg.f, serv_envolvidos, ind_inst, ind_cli, 0);
    N_CONFIRM := 0;
    erro_oper := false; { usado para fechar o arquivo }
    oper := CLOSING_FILE;
 end;

from FECHANDO to ESPERA
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv, nro_seq)
{ Ha' mais de um Cliente com o arquivo aberto e o MetaServ confirmou realizacao
  da operacao }
provided (oper = REG_CLOSING_FILE) and (status = ARQ_FECHADO)
priority ALTA
begin
   { operacao encerrada }
   output A.confirm_gfs_close(fid);
end;


from FECHANDO to ESPERA
when B.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv, nro_seq)
provided (oper = REG_CLOSING_FILE) and (status = ERRO)
priority ALTA
begin
   { falha ao notificar meta-serv }
   output A.confirm_gfs_close(-2);
end;



{ transicoes para o arquivo aberto ------------------------------------- }
```

```
 from ABERTO to FECHANDO  { Cliente solicita fechar arquivo de forma assincrona }
 when A.gfs_close(fid, espera)
 provided (not espera)
 begin
    output A.confirm_gfs_close(fid);
    N_CONFIRM := 0;
    RESP := false;
    oper := REG_CLOSING_FILE;
    InverteConfServ(serv_envolvidos);
    { registra operacao fechando no meta-servidor }
    output B.reg_file_status(msg.f, ARQ_FECHANDO, ind_inst, ind_cli,
                             hash(msg.arq), 0);
    erro_oper := false
 end;

 from ABERTO to FECHANDO   { Cliente solicita fechar arquivo de forma sincrona }
 when A.gfs_close(fid, espera)
 provided (espera)
 begin
    N_CONFIRM := 0;
    RESP := true;
    InverteConfServ(serv_envolvidos);
    oper := REG_CLOSING_FILE;
    {output A.p_close(fid, serv_envolvidos, 0);}
    output B.reg_file_status(msg.f, ARQ_FECHANDO, ind_inst, ind_cli,
                             hash(msg.arq), 0);
    erro_oper := false;
 end;


 from ABERTO to LENDO  { chega pedido para ler o arquivo }
 when A.gfs_read(fid, buf, count)
 begin
    msg.f := fid; { armazena o fid para retransmissao }
    msg.buf := buf; { usado para a recepcao de dados apos a leitura }
    msg.count := count; {usado para comparacao de primitivas }
    { NIL sinaliza que <buf> sera' usado para armazenar os bytes apos a leitura }
    DetUnidadesEnvolvidas(buf, count, lista_s, N_BLOCK);
    N_CONFIRM := 0;
    N_BYTES := 0;
    N_RETRANS := 0;
    erro_oper := false;
    output B.gfs_sread(fid, lista_s, ind_cli, 0)
end;

from ABERTO to ESCREVENDO  { chega pedido para escrever no arquivo }
when A.gfs_write(fid, buf, count)
begin
   msg.f := fid;
   msg.buf := buf; { usado para a comparacao de primitivas }
   msg.count := count; { usado para a comparacao de primitivas }
   { buf <> NIL sinaliza que <buf> sera' usado para obter os dados a serem
     escritos nos servidores }
   DetUnidadesEnvolvidas(buf, count, lista_s, N_BLOCK);
   N_CONFIRM := 0;
   N_BYTES := 0;
   N_RETRANS := 0;
   erro_oper := false;
   output B.gfs_swrite(fid, lista_s, ind_cli, 0)
end;

from ABERTO to same  { chega pedido para ajustar posicao do arquivo }
when A.gfs_lseek(fid, offset, origem)
begin
   output A.confirm_gfs_lseek(fid, AjustarPosicaoArquivo(fid,lista_s))
end;
```

```
{ ------------------------------------------------------------------------ }


{ inicio da fase de leitura do arquivo }

{ servidores comecam a responder ao pedido de leitura retornando os blocos de
  dados solicitados }
from LENDO to same
when B.confirmdado(fid, ind_serv, dados, status, nro_seq)
provided N_CONFIRM+1 < N_BLOCK
priority ALTA
begin
   { verifica se o servidor ja' foi confirmado -> retransmissao }
   if not VerificaServConfirm(ind_serv) then
   begin
      AlteraServConfirmES(ind_serv, lista_s);      { confirma servidor }
      CopiaDadosBuffer(msg.buf, dados);
      if status = ERRO then erro_oper := true;
      N_BYTES := N_BYTES + AvaliaNroBytes(dados);
      N_CONFIRM := N_CONFIRM + 1;
  end
end;

{ todos os servidores responderam -> operacao de leitura teve sucesso }
from LENDO to ABERTO
when B.confirmdado(fid, ind_serv, dados, status, nro_seq)
{ servidor nao havia sido confirmado }
provided (N_CONFIRM+1 = N_BLOCK) and (not VerificaServConfirm(ind_serv))
priority ALTA
begin
   AlteraServConfirmES(ind_serv, lista_s);      { confirma servidor }
   CopiaDadosBuffer(msg.buf, dados);
   N_BYTES := N_BYTES + AvaliaNroBytes(dados);
   if (erro_oper) or (status = ERRO) then status := ERRO;
   output A.respread(fid, msg.buf, N_BYTES, status);
   RestServEnvolvidos;
end;

from LENDO to same
when B.confirmdado(fid, ind_serv, dados, status, nro_seq)
{ servidor ja' havia sido confirmado }
provided (N_CONFIRM+1 = N_BLOCK) and (VerificaServConfirm(ind_serv))
priority ALTA
begin   { retransmissao de mensagem }
end;

from LENDO to same
provided N_RETRANS < MAX_RETRANS
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   { retransmite p_read aos servidores que ainda nao responderam }
   output B.gfs_sread(msg.f, lista_s, ind_cli, 0)
end;

from LENDO to ABERTO
provided N_RETRANS = MAX_RETRANS
priority BAIXA
delay(T)
begin
   output A.respread(msg.f, msg.buf, 0, ERRO);
   RestServEnvolvidos;
end;

{ inicio da fase de escrita }
```

```
{ servidores comecam a enviar confirmacao da operacao de escrita }
from ESCREVENDO to same
when B.confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
provided N_CONFIRM+1 < N_BLOCK
priority ALTA
begin
   { verifica se o servidor ja' foi confirmado -> retransmissao }
   if not VerificaServConfirm(ind_serv) then
   begin
      AlteraServConfirmES(ind_serv, lista_s);      { confirma servidor }
      N_BYTES := N_BYTES + AvaliaNroBytes(DadosServ(ind_serv));
      N_CONFIRM := N_CONFIRM + 1;
      if status = ERRO then erro_oper := true;
   end
end;

{ todos os servidores envolvidos com o arquivo corrente respondem ao pedido de
  escrita }
from ESCREVENDO to ABERTO
when B.confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
provided (N_CONFIRM+1 = N_BLOCK) and (not VerificaServConfirm(ind_serv))
priority ALTA
begin
   AlteraServConfirmES(ind_serv, lista_s);      { confirma servidor }
   N_BYTES := N_BYTES + AvaliaNroBytes(DadosServ(ind_serv));
   if (erro_oper) or (status = ERRO) then status := ERRO;
   output A.respwrite(fid, N_BYTES, status);
   RestServEnvolvidos;
end;

from ESCREVENDO to same
when B.confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
provided (N_CONFIRM+1 = N_BLOCK) and (VerificaServConfirm(ind_serv))
priority ALTA
begin   { retransmissao da mensagem }
end;

from ESCREVENDO to same
provided N_RETRANS < MAX_RETRANS
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   { retransmite p_write aos servidores que ainda nao responderam }
   output B.gfs_swrite(msg.f, lista_s, ind_cli, 0)
end;

from ESCREVENDO to ABERTO
provided N_RETRANS = MAX_RETRANS
priority BAIXA
delay(T)
begin
   output A.respwrite(msg.f, 0, ERRO);
   RestServEnvolvidos
end;

{  1. Remocao de arquivo distribuido ---------------------------------------- }

  from ESPERA to REMOVENDO   { # 51 }
  when A.gfs_unlink(arq, espera)   { Cliente solicita remocao do arquivo }
  begin
     N_RETRANS := 0;
     if (espera) then  { servico sincrono }
        RESP := true
     else
     begin             { servico assincrono }
        output A.confirm_gfs_unlink(arq, OK);
        RESP := false;
```

```
   end;
   oper := REG_UNLINKING;
   msg.arq := arq;
   output B.reg_file_status_name(arq, ARQ_REMOVENDO, ind_inst, ind_cli,
                                 hash(arq), 0); { altera status no meta-servidor }
end;

{  2. Entrando em contato com o meta-servidor para iniciar a operacao ------ }

 from REMOVENDO to same
 provided (N_RETRANS < MAX_RETRANS) and (oper = REG_UNLINKING)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    output B.reg_file_status_name(msg.arq, ARQ_REMOVENDO, ind_inst, ind_cli,
                                  hash(msg.arq), 1);
 end;

 from REMOVENDO to same
 provided (N_RETRANS = MAX_RETRANS) and (oper = REG_UNLINKING)
 priority BAIXA
 delay(T)
 begin
    if (RESP) then
       output A.confirm_gfs_unlink(msg.arq, ERRO);
 end;

from REMOVENDO to same
when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                       nro_seq)   { meta-servidor responde }
provided (status = OK)  and (oper = REG_UNLINKING) { exito ao registrar }
begin
   N_CONFIRM := 0;
   N_RETRANS := 0;
   oper := UNLINKING;
   msg.f := fid;
   { armazena os servidores envolvidos }
   IniciaServEnvolvidos(lista_serv, serv_envolvidos);
   ArmazenaServEnvolvidos(lista_serv);
   N_SERVERS := NroServEnvolvidos;
   output B.gfs_sunlink(msg.f, serv_envolvidos, ind_inst, ind_cli, 0);
end;

from REMOVENDO to ESPERA
when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                       nro_seq)   { meta-servidor responde }
provided (status <> OK) and (oper = REG_UNLINKING)  { falha ao registrar }
begin
   if (RESP) then
      output A.confirm_gfs_unlink(msg.arq, ERRO);
end;

{ 3. Entrando em contato com os Servidores ------------------------------- }

from REMOVENDO to same
when A.gfs_unlink(arq, espera)   { Cliente solicita remocao do arquivo }
begin
   { retransmissao de pedido -> ignorar pedido }
end;

from REMOVENDO to same
{ todos os servidores responderam `as solicitacoes do Mestre }
when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
{ nao e' uma retransmissao }
provided (N_CONFIRM+1 = N_SERVERS) and (not VerificaServConfirm(ind_serv))
priority ALTA
begin
```

```
   { atualiza confirmacao do servidor }
   AlteraServConfirm(ind_serv, serv_envolvidos);
   { verifica se ocorreu erro em alguma confirmacao do servidor }
   if erro_oper then status := ERRO;
   N_RETRANS := 0;
   oper := REG_UNLINKING_F;
   output B.reg_file_status_name(msg.arq, ARQ_REMOVIDO, ind_inst, ind_cli,
                          hash(msg.arq), 0); { altera status no meta-servidor }
end;

from REMOVENDO to same
{ todos os servidores responderam `as solicitacoes do Cliente }
when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
{  e' uma retransmissao }
provided (N_CONFIRM+1 = N_SERVERS) and (VerificaServConfirm(ind_serv))
priority ALTA
begin   { ignora mensagem }
end;

from REMOVENDO to same
{ um servidor respondeu ao Mestre }
when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = OK) and (N_CONFIRM+1 < N_SERVERS)
priority ALTA
begin
   if not VerificaServConfirm(ind_serv) then
   begin
      N_CONFIRM := N_CONFIRM + 1;
      { atualiza confirmacao do servidor }
      AlteraServConfirm(ind_serv, serv_envolvidos);
   end
end;

from REMOVENDO to same
{ um servidor respondeu ao Cliente }
when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
provided (status = ERRO) and (N_CONFIRM+1 < N_SERVERS)
priority ALTA
begin
   if not VerificaServConfirm(ind_serv) then
   begin
      { atualiza confirmacao do servidor }
      AlteraServConfirm(ind_serv, serv_envolvidos);
      N_CONFIRM := N_CONFIRM + 1;
      erro_oper := true;
   end
end;

from REMOVENDO to same
provided (N_RETRANS < MAX_RETRANS) and (oper = UNLINKING)
priority BAIXA
delay(T)
begin
   N_RETRANS := N_RETRANS+1;
   { retransmite pedido aos clientes que ainda nao responderam }
   output B.gfs_sunlink(msg.f, serv_envolvidos, ind_inst, ind_cli, 0);
end;

from REMOVENDO to same
provided (N_RETRANS = MAX_RETRANS) and (oper = UNLINKING)
priority BAIXA
delay(T)
begin
   if (RESP) then
      output A.confirm_gfs_unlink(msg.arq, ERRO);
   N_RETRANS := 0;
   oper := REG_UNLINKING_F;
   output B.reg_file_status_name(msg.arq, ARQ_REMOVIDO, ind_inst, ind_cli,
```

```
                              hash(msg.arq), 1); { altera status no meta-servidor }
   end;


  { 4. Entrando em contato com o meta-servidor para finalizar a operacao ------ }

    from REMOVENDO to same
    provided (N_RETRANS < MAX_RETRANS) and (oper = REG_UNLINKING_F)
    priority BAIXA
    delay(T)
    begin
       N_RETRANS := N_RETRANS+1;
       output B.reg_file_status_name(msg.arq, ARQ_REMOVIDO, ind_inst, ind_cli,
                                     hash(msg.arq), 1);
    end;

    from REMOVENDO to same
    provided (N_RETRANS = MAX_RETRANS) and (oper = REG_UNLINKING_F)
    priority BAIXA
    delay(T)
    begin
       if (RESP) then
          output A.confirm_gfs_unlink(msg.arq, ERRO);
    end;

   from REMOVENDO to ESPERA
   when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                          nro_seq)   { meta-servidor responde }
   provided (status = OK)  and (oper = REG_UNLINKING_F)  { exito ao registrar }
   begin
      if (RESP) then
         if (erro_oper) then
            output A.confirm_gfs_unlink(msg.arq, ERRO)
         else
            { operacao encerrada com sucesso }
            output A.confirm_gfs_unlink(msg.arq, OK);
   end;

   from REMOVENDO to ESPERA
   when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                          nro_seq)   { meta-servidor responde }
   provided (status <> OK)  and (oper = REG_UNLINKING_F)  { exito ao registrar }
   begin
            output A.confirm_gfs_unlink(msg.arq, ERRO)
   end;


{ ------------------------------------------------------------------------- }

{ 1. Renomeando arquivo distribuido ------------------------------------- }


   from ESPERA to RENOMEANDO
   { Cliente solicita renomear arquivo }
   when A.gfs_rename(arq_velho, arq_novo, espera)
   begin
      N_RETRANS := 0;
      if (espera) then     { servico sincrono }
         RESP := true
      else begin           { servico assincrono }
         output A.confirm_gfs_rename(arq_velho, OK, false);
         RESP := false;
      end;
      oper := REG_RENAMING;
      msg.arq := arq_velho;
      msg.arq2 := arq_novo;  { armazena nome dos arquivos para retransmissoes }
      output B.reg_file_status_name(arq_velho, ARQ_RENOMEANDO, ind_inst, ind_cli,
                        hash(arq_velho), 0); { altera status no meta-servidor }
```

```
end;

{  2. Entrando em contato com o meta-servidor para iniciar a operacao ---- }

 from RENOMEANDO to same
 provided (N_RETRANS < MAX_RETRANS) and (oper = REG_RENAMING)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    output B.reg_file_status_name(msg.arq, ARQ_RENOMEANDO, ind_inst, ind_cli,
                                  hash(msg.arq), 1);
 end;

 from RENOMEANDO to ESPERA
 provided (N_RETRANS = MAX_RETRANS) and (oper = REG_RENAMING)
 priority BAIXA
 delay(T)
 begin
    if (RESP) then
       output A.confirm_gfs_rename(msg.arq, ERRO, true);
 end;

from RENOMEANDO to same
when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                       nro_seq)   { meta-servidor responde }
provided (status = OK) and (oper <> UN_REG_RENAMING)   { exito ao registrar }
begin
   msg.f := fid; { armazena fid do arquivo }
   N_CONFIRM := 0;
   IniciaServEnvolvidos(lista_serv, serv_envolvidos);
   { armazena os servidores envolvidos }
   ArmazenaServEnvolvidos(lista_serv);
   CopiaVet(lista_serv);
   N_SERVERS := NroServEnvolvidos;
   { entra em contato com o novo meta-servidor, ainda que seja o mesmo que o
     primeiro }
   oper := CONTACTING_SEC_MS;
   { passa os metadados do arquivo ao segundo meta-serv }
   output B.reg_file_status_ren(msg.arq2, lista_serv, tam_unid, ind_inst,
                                ind_cli, hash(msg.arq2), 0);
end;

from RENOMEANDO to ESPERA
when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                       nro_seq)   { meta-servidor responde }
provided (status <> OK)   { falha ao registrar }
begin
   if (RESP) then
      output A.confirm_gfs_rename(msg.arq, ERRO, true);
end;   { fim da operacao }


{  3. Entrando em contato com o segundo meta-servidor para registrar a operacao }

 from RENOMEANDO to same
 provided (N_RETRANS < MAX_RETRANS) and (oper = CONTACTING_SEC_MS)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    RecuperaListaServ(lista_aux);
    output B.reg_file_status_ren(msg.arq2, lista_aux, msg.tam, ind_inst,
                                 ind_cli, hash(msg.arq2), 1);
 end;

 from RENOMEANDO to same
 provided (N_RETRANS = MAX_RETRANS) and (oper = CONTACTING_SEC_MS)
```

```
 { falha ao contactar o segundo meta-servidor }
 priority BAIXA
 delay(T)
 begin
    if (RESP) then
       output A.confirm_gfs_rename(msg.arq, ERRO, false);
   { restaura status no primeiro meta-servidor }
   oper := UN_REG_RENAMING;
   N_RETRANS := 0;
   output B.reg_file_status_name(msg.arq, ARQ_FECHADO, ind_inst, ind_cli,
                         hash(msg.arq), 1); { restaura status no meta-servidor }
 end;

from RENOMEANDO to same
{ segundo meta-servidor responde }
when B.confirm_reg_na(fid, status, ind_inst, ind_cli, nro_seq)
provided (status = OK)    { exito ao registrar }
begin
   N_CONFIRM := 0;
   N_RETRANS := 0;
   oper := RENAMING;
   msg.f2 := fid;  { armazena o fid do segindo meta-servidor }
   output B.gfs_srename(msg.f, msg.arq, msg.arq2, serv_envolvidos, ind_inst,
                        ind_cli, 0);
end;

from RENOMEANDO to same
{ segundo meta-servidor responde }
when B.confirm_reg_na(fid, status, ind_inst, ind_cli, nro_seq)
provided (status <> OK)   { falha ao registrar }
begin
   oper := UN_REG_RENAMING;
   N_RETRANS := 0;
   output B.reg_file_status_name(msg.arq, ARQ_FECHADO, ind_inst, ind_cli,
                hash(msg.arq), 0); { restaura status no primeiro meta-servidor }
end;

 from RENOMEANDO to same
 { retransmitindo pedido para que o primeiro meta-serv restabeleca o status do
   arquivo que iria ser renomeado }
 provided (N_RETRANS < MAX_RETRANS) and (oper = UN_REG_RENAMING)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    output B.reg_file_status_name(msg.arq, ARQ_FECHADO, ind_inst, ind_cli,
                                  hash(msg.arq), 1);
 end;

 from RENOMEANDO to ESPERA
 { falha ao contactar primeiro meta-serv para que o status do arquivo que iria
   ser renomeado fosse restaurado }
 provided (N_RETRANS = MAX_RETRANS) and (oper = UN_REG_RENAMING)
 priority BAIXA
 delay(T)
 begin
    if (RESP) then
       output A.confirm_gfs_rename(msg.arq, ERRO, true);
 end;

from RENOMEANDO to ESPERA
when B.confirm_reg_arq(fid {arq}, status, tam_unid, lista_serv, ind_inst,
   ind_cli, nro_seq)   { primeiro meta-servidor responde: status restabelecido }
provided (oper = UN_REG_RENAMING)
begin
   if (RESP) then
       output A.confirm_gfs_rename(msg.arq, ERRO, true);
end;
```

```
{  4. Solicitando renomeacao aos servidores ---------------------------------- }


  from RENOMEANDO to same
  { Cliente solicita renomear arquivo com operacao ja' em andamento }
  when A.gfs_rename(arq_velho, arq_novo, espera)
  begin  { ignora solicitacao }
  end;

  from RENOMEANDO to same
  { todos os servidores responderam `as solicitacoes do Cliente}
  when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
  { nao e' uma retransmissao }
  provided (N_CONFIRM+1 = N_SERVERS) and (not VerificaServConfirm(ind_serv))
  priority ALTA
  begin
     { atualiza confirmacao do servidor }
     AlteraServConfirm(ind_serv, serv_envolvidos);
     N_RETRANS := 0;
     { status indica que ultimo servidor a responder retornou erro }
     if (erro_oper) or (status = ERRO) then
     begin
        { Restaura status fechado no segundo meta_serv (elimina arquivo que seria
          criado durante a renomeacao, o que eh feito enviando ao segundo
          metaservidor uma msg com o status ARQ_REMOVIDO }
        oper := UN_REG_RENAMING_2;
        {  a funcao hash determina o endereco do segundo meta-serv }
        output B.reg_file_status_name(msg.arq2, ARQ_REMOVIDO, ind_inst, ind_cli,
                                      hash(msg.arq2), 0);
     end
     else
     begin
        { Confirma exito da realizacao da operacao nos dois meta-servidores}
        oper := CONFIRM_META_2;
        output B.reg_file_status_rename(msg.arq2, msg.f, ARQ_RENOMEADO, ind_inst,
           ind_cli, hash(msg.arq2), 0); { primeiro notificar o segundo meta-serv }
     end
  end;

  from RENOMEANDO to same
  { todos os servidores responderam `as solicitacoes do Cliente }
  when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
  {  e' uma retransmissao }
  provided (N_CONFIRM+1 = N_SERVERS) and (VerificaServConfirm(ind_serv))
  priority ALTA
  begin   { ignora mensagem }
  end;

  from RENOMEANDO to same
  { um servidor respondeu ao Cliente }
  when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
  provided (status = OK) and (N_CONFIRM+1 < N_SERVERS)
  priority ALTA
  begin
     if not VerificaServConfirm(ind_serv) then
     begin
        N_CONFIRM := N_CONFIRM + 1;
        { atualiza confirmacao do servidor }
        AlteraServConfirm(ind_serv, serv_envolvidos);
     end
  end;

  from RENOMEANDO to same
  { um servidor respondeu ao Cliente com erro }
  when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
  provided (status = ERRO) and (N_CONFIRM+1 < N_SERVERS)
```

```
 priority ALTA
 begin
    if not VerificaServConfirm(ind_serv) then
    begin
       { atualiza confirmacao do servidor }
       AlteraServConfirm(ind_serv, serv_envolvidos);
       N_CONFIRM := N_CONFIRM + 1;
       erro_oper := true;
    end
 end;


{ erro ao renomear arquivo (falha ou erro dos servidores); notificando o segundo
  meta-serv }

  from RENOMEANDO to same
  { retransmitindo pedido para o segundo meta-serv }
  provided (N_RETRANS < MAX_RETRANS) and ((oper = UN_REG_RENAMING_2) or
           (oper = CONFIRM_META_2))
  priority BAIXA
  delay(T)
  begin
     N_RETRANS := N_RETRANS+1;
     case oper of
        {  a funcao hash determina o endereco do segundo meta-serv }
        UN_REG_RENAMING_2: output B.reg_file_status_name(msg.arq2, ARQ_FECHADO,
                                    ind_inst, ind_cli, hash(msg.arq2), 1);
        { Todos os servidores confirmaram com sucesso a renomeacao do arquivo;
          alertando segundo meta_serv }
        CONFIRM_META_2:output B.reg_file_status_rename(msg.arq2, msg.f2,
                             ARQ_RENOMEADO, ind_inst, ind_cli, hash(msg.arq2), 1);
     end;
  end;

  from RENOMEANDO to same
  { falha ao contactar primeiro meta-serv para que o status do arquivo que iria
    ser renomeado fosse restaurado }
  provided (N_RETRANS = MAX_RETRANS) and ((oper = UN_REG_RENAMING_2) or
           (oper = CONFIRM_META_2))
  priority BAIXA
  delay(T)
  begin
     { independente se a requisicao ao segundo meta-serv foi bem ou mal sucedido,
       o primeiro meta-serv deve ser contatado para restaurar seu status }
     N_RETRANS := 0;
     if (oper = UN_REG_RENAMING_2) then
        { reaproveita transicoes para o tratamento de retransmissoes }
        oper := UN_REG_RENAMING
     else
     begin
        oper := CONFIRM_META_1;
        erro_oper := true;  { falha do meta-serv2 }
     end;
     { endereco do primeiro meta-serv determinado pela funcao hash }
     output B.reg_file_status_name(msg.arq, ARQ_FECHADO, ind_inst, ind_cli,
                                   hash(msg.arq), 1);
  end;

  from RENOMEANDO to same
  { segundo meta-ser respondeu ao pedido de alteracao de status }
  when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                         nro_seq)
  provided (oper = UN_REG_RENAMING_2) or (oper = CONFIRM_META_2)
  priority ALTA
  begin
     { independente se a requisicao ao segundo meta-serv foi bem ou mal sucedido,
       o primeiro meta-serv deve ser contatado para restaurar seu status }
     N_RETRANS := 0;
```

```
     if (oper = UN_REG_RENAMING_2) then
     begin
        { reaproveita transicoes para o tratamento de retransmissoes }
        oper := UN_REG_RENAMING;
        { endereco do primeiro meta-serv determinado pela funcao hash }
        output B.reg_file_status_name(msg.arq, ARQ_FECHADO, ind_inst, ind_cli,
                                      hash(msg.arq),0);
     end
     else begin
        { contacta meta-serv1 para registrar resultado da operacao }
        oper := CONFIRM_META_1;
        if status <> ERRO then
           output B.reg_file_status_rename(msg.arq, msg.f2, RENOMEADO_M1,
                                           ind_inst, ind_cli, hash(msg.arq), 0)
        else begin{houve erro ao se registrar a operacao no segundo metaservidor}
           erro_oper := true;
           output B.reg_file_status_rename(msg.arq, msg.f2, RENOMEADO_M1,
                                          ind_inst, ind_cli, hash(msg.arq), 0);
        end;
     end;
  end;

{ Todos os servidores confirmaram com sucesso a renomeacao do arquivo; alertando
  primeiro meta_serv }

 from RENOMEANDO to same
 provided (N_RETRANS < MAX_RETRANS) and (oper = CONFIRM_META_1)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    output B.reg_file_status_rename(msg.arq, msg.f2, RENOMEADO_M1, ind_inst,
                                    ind_cli, hash(msg.arq), 1);
 end;

 from RENOMEANDO to ESPERA
 { falha ao contactar um ou mais servidores }
 provided (N_RETRANS = MAX_RETRANS) and (oper = CONFIRM_META_1)
 priority BAIXA
 delay(T)
 begin
    output A.confirm_gfs_rename(msg.arq, ERRO, false)
 end;

 from RENOMEANDO to ESPERA
 when B.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                        nro_seq)
 provided (oper = CONFIRM_META_1)
 priority ALTA
 begin
    if (RESP) then
       if (erro_oper) then
          output A.confirm_gfs_rename(msg.arq, ERRO, true)
       else
          output A.confirm_gfs_rename(msg.arq, OK, true)
 end; { fim da operacao }

{  Retransmissao aos servidores para que renomeiem seus fragmentos locais do
   arquivo distribuido }

 from RENOMEANDO to same
 { retransmite pedido aos servidores que ainda nao responderam }
 provided (N_RETRANS < MAX_RETRANS) and (oper = RENAMING)
 priority BAIXA
 delay(T)
 begin
    N_RETRANS := N_RETRANS+1;
    output B.gfs_srename(msg.f, msg.arq, msg.arq2, serv_envolvidos, ind_inst,
```

```
                          ind_cli, 0);
  end;

  from RENOMEANDO to same
  { falha ao contactar um ou mais servidores }
  provided (N_RETRANS = MAX_RETRANS) and (oper = RENAMING)
  priority BAIXA
  delay(T)
  begin
    { entra em contato com metaservidores para desfazer a operacao -> situacao
      similar a outra(s) }
    oper := UN_REG_RENAMING_2;
    output B.reg_file_status_name(msg.arq2, ARQ_REMOVIDO, ind_inst, ind_cli,
                                  hash(msg.arq2), 1);
  end;


{ mensagens que devem ser ignoradas }

        from IgnoraConfirmOpen to same
        when B.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
        begin
        end;

        from IgnoraConfirmClose to same
        when B.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
        begin
        end;

        from IgnoraConfirmRead to same
        when B.confirmdado(fid, ind_serv, dados, status, nro_seq)
        begin
        end;

        from IgnoraConfirmWrite to same
        when B.confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
        begin
        end;

        from IgnoraConfirmUnlink to same
        when B.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
        begin
        end;



  end; { fim da definicao do corpo de InsCli }

{ tipos e variaveis do cliente }

type
   info = record
          arq:TipoArquivo;     { path do arquivo }
          acesso:TipoOperacao; { operacao atual sobre o arquivo }
          fid:fd;              { fid do arquivo }
   end;
   TabArq = array[1..Cli] of info;

var
   TabArquivos: TabArq;         { tabela de arquivos do cliente }
   indice, k: integer;
   string_vazia: TipoArquivo;
   { armazena o ultimo identificador de mensagem enviado por uma Instancia do
     Cliente }
   nro_msg: array[1..Cli] of TipoNumeracaoMsg;
   nro_sequencia:TipoNumeracaoMsg;

modvar
```

```
   Instancia_Cliente: array[1..Cli] of InstCli;

state CONTROLANDO;

{ funcoes do Cliente ------------------------------------------------------ }

procedure InsereArquivoTab(arquivo:TipoArquivo; oper:TipoOperacao;
                           var TabCli:TabArq; var indice:integer);
{ insere um arquivo na Tabela de Arquivos e devolve o indice onde foi armazenado }
var i: integer;
begin
   i := 1;
   while (i <= Cli) and (TabCli[i].acesso <> DESOCUPADO) do
      i := i+1;
   if i <= Cli then              { foi encontrada uma posicao vaga na tabela }
   begin
      TabCli[i].arq := arquivo;
      TabCli[i].acesso := oper;
      indice := i;     { retorna o indice do arquivo inserido }
   end
   else
      indice := IndiceInvalido; { sem espaco na tabela }
 end; { InsereArquivoTab }

procedure InsereFidTab(f:fd; pos:integer; var TabCli:TabArq);
{ insere o fid de um arquivo na posicao <pos> da Tabela de Arquivos}
begin
   TabCli[pos].fid := f
end; { InsereFidTab }

procedure RetiraArquivoTab(pos:integer; var TabCli:TabArq);
{ retira um arquivo localizado na posicao <pos> da Tabela de Arquivos }
var i:integer;
begin
   with TabCli[pos] do begin
      acesso := DESOCUPADO;
      fid := -1; { fid invalido }
      for i:=1 to MAXCOMPR do
         arq[i] := ' '
   end
end; { RetiraArquivoTab }

function cmpstr(str1, str2:TipoArquivo):boolean;
{ verifica se duas strings sao iguais }
var i:integer;
begin
   i := 1;
   while (i <= MAXCOMPR) and (str1[i] = str2[i]) do
      i := i+1;
   if i > MAXCOMPR then
      cmpstr := true
   else
      cmpstr := false
end; { cmpstr }

function VerificaArquivoTab(arquivo:TipoArquivo; f:fd):integer;
{ verifica se um arquivo (dado seu nome ou seu fid) esta` inserido na Tabela,
  retornando seu indice }
var i:integer;
begin
   i := 1;
   if f >= 0 then             { vefifica se o fid e' um numero valido }
   begin
      while (i <= Cli) and (TabArquivos[i].fid <> f) do
         i := i+1;
      if i <= Cli then            { arquivo foi encontrado }
         VerificaArquivoTab := i
      else
```

```
         VerificaArquivoTab := IndiceInvalido;  { o arquivo nao foi encontrado }
   end
   else begin    { pesquisa e' feita pelo nome do arquivo }
      while (i <= Cli) and (not cmpstr(TabArquivos[i].arq, arquivo)) do
         i := i+1;
      if i <= Cli then            { arquivo foi encontrado }
         VerificaArquivoTab := i
      else
         VerificaArquivoTab := IndiceInvalido;  { o arquivo nao foi encontrado }
   end
end; { VerificaArquivoTab }

procedure IniciaTabela(var TabCli:TabArq);
{ inicia a Tabela de Arquivos do Cliente }
var i,j:integer;
begin
   for i:=1 to Cli do
   begin
      with TabCli[i] do begin
         acesso := DESOCUPADO;   { lugar desocupado na tabela }
         fid := -1;    { fid invalido }
         for j:=1 to MAXCOMPR do
            arq[j] := ' '; { string vazia }
      end;
   end
end; { IniciaTabela }

function PathValido(arq:TipoArquivo):boolean;
{ verifica se um path esta' sintaticamente correto; retorna <true> se estiver; caso
  contrario, retorna <false> }
{primitive;}
begin
   PathValido := true  { valor de retorno default para a simulacao }
end;

procedure GeraNroSeq(indice:integer; var k:integer);
{ primitive }
begin
   k := k+1;
   nro_msg[indice] := k;
   if k = NRO_MAX_SEQ then
      k := -1 { reinicia numeracao }
end; { GeraNroSeq }

function RecuperaNroSeq(indice:integer):TipoNumeracaoMsg;
{ primitive }
begin
   RecuperaNroSeq := nro_msg[indice]
end; { RecuperaNroSeq }

function ComparaNroSeq(indice1, indice2:integer):boolean;
{ primitive }
begin
   ComparaNroSeq := (nro_msg[indice1] = indice2)
end; { ComparaNroSeq }

procedure IniciaSeqMsg;
begin
   k := -1
end; { IniciaSeqMsg }

function ListaServValida(x:serv_list):boolean;
{ Verifica se uma lista de servidores nao possui elementos repetidos, elementos
  negativos e se possui um terminador (-1) }
var i,j:integer;
    ret, igual:boolean;
begin
   if x[1] = -1 then ListaServValida := true
```

```
   else begin
      i := 1;
      ret := false;
      igual := false;
      while (i <= Compr_lista_serv) and (x[i] <> -1) and (not ret) and (not igual)
      do begin
         if (x[i] <= 0) then  { verifica se um dos elementos do vetor e' negativo }
            ret := true
         else begin { verifica se nao ha' outro elemento igual no vetor }
            j := i+1;
            while (j <= Compr_lista_serv) do begin
                if (x[j] <> -1) and (not igual) then
                   if x[i] = x[j] then
                      igual := true;
               j := j+1;
            end;
            i := i+1;
         end;
      end;
      if (i <= Compr_lista_serv) and (not ret) and (not igual) then
         ListaServValida := true
      else
         ListaServValida := false;
    end
end; { ListaServValida }


{ inicializacao }

initialize to CONTROLANDO
begin
   IniciaTabela(TabArquivos); { inicia a tabela de arquivos do cliente }
   IniciaSeqMsg;              { inicia  o enumerador de mensagens }
   string_vazia[1] := ' ';
end;

{ comunicacao entre os Usuarios e as instancias InsCli do Cliente --------- }

trans
   from CONTROLANDO to same
   { usuario solicita abertura de arquivo }
   when C.gfs_open(arq, modo, tam_unid, lista_s)
   { PATH e lista de servidores validos }
   provided (PathValido(arq)) and (ListaServValida(lista_s))
   priority ALTA
   begin
      indice := VerificaArquivoTab(arq, -1);   { verifica se o arquivo ja' existe }
      if indice <> IndiceInvalido then {o arquivo ja' exite na Tabela de Arquivos}
         case TabArquivos[indice].acesso of
           { o arquivo ja' esta' aberto -> confirma arquivo aberto para o Usuario }
            ABERTO, LENDO, ESCREVENDO:
                                output C.confirm_gfs_open(TabArquivos[indice].fid);
            ABRINDO: begin {ignora popen -> retransmissao} end;
            { a instancia para o arquivo existe e esta' sendo realizada uma
              operacao UNLINK, RENAME ou CLOSE}
            RENOMEANDO, REMOVENDO, FECHANDO: output C.confirm_gfs_open(-1);
          end
      else begin            { sera' criada uma instancia para esse arquivo }
         InsereArquivoTab(arq, ABRINDO, TabArquivos, indice);
         GeraNroSeq(indice, k);
         init Instancia_Cliente[indice] with CorpoInstCli(indice, ind_cli);
         connect MA[indice] to Instancia_Cliente[indice].A;
         connect MB[indice] to Instancia_Cliente[indice].B;
         output MA[indice].gfs_open(arq, modo, tam_unid, lista_s);
      end
   end;

   when C.gfs_open(arq, modo, tam_unid, lista_s)
```

```
provided not (PathValido(arq) and ListaServValida(lista_s))  { PATH invalido }
priority ALTA
begin
   if not PathValido(arq) then
      output C.confirm_gfs_open(-1{, PATH_INVALIDO})
   else
      output C.confirm_gfs_open(-1{, LISTA_DE_SERVIDORES_INVALIDA})
end;

when C.gfs_unlink(arq, espera)  { usuario solicita remocao de arquivo }
provided PathValido(arq)        { PATH valido }
priority ALTA
begin
   indice := VerificaArquivoTab(arq, -1);   { verifica se o arquivo ja' existe }
   if indice <> IndiceInvalido then
      begin
         if TabArquivos[indice].acesso <> REMOVENDO then
            { esta' sendo realizada outra operacao sobre o arquivo }
            output C.confirm_gfs_unlink(arq, ERRO_UNLINK)
      end
   else begin     { sera' criada uma instancia para esse arquivo }
      InsereArquivoTab(arq, REMOVENDO, TabArquivos, indice);
      GeraNroSeq(indice, k);
      init Instancia_Cliente[indice] with CorpoInstCli(indice, ind_cli);
      connect MA[indice] to Instancia_Cliente[indice].A;
      connect MB[indice] to Instancia_Cliente[indice].B;
      output MA[indice].gfs_unlink(arq, espera);
   end;
end;

when C.gfs_unlink(arq, espera)
provided not PathValido(arq)
priority ALTA
begin
   output C.confirm_gfs_unlink(arq, PATH_INVALIDO)
end;

{ usuario solicita renomear arquivo }
when C.gfs_rename(arq_velho, arq_novo, espera)
provided (PathValido(arq_velho)) and (PathValido(arq_novo)) and
         (not cmpstr(arq_velho, arq_novo))
priority ALTA
begin
   { verifica se o arquivo ja' existe }
   indice := VerificaArquivoTab(arq_velho, -1);
   if indice <> IndiceInvalido then
      begin
         if TabArquivos[indice].acesso <> RENOMEANDO then
            output C.confirm_gfs_rename(arq_velho, ERRO_RENAME, false)
      end
   else begin           { sera' criada uma instancia para esse arquivo }
      InsereArquivoTab(arq_velho, RENOMEANDO, TabArquivos, indice);
      GeraNroSeq(indice, k);
      init Instancia_Cliente[indice] with CorpoInstCli(indice, ind_cli);
      connect MA[indice] to Instancia_Cliente[indice].A;
      connect MB[indice] to Instancia_Cliente[indice].B;
      output MA[indice].gfs_rename(arq_velho, arq_novo, espera);
   end
end;

when C.gfs_rename(arq_velho, arq_novo, espera)
provided ((not PathValido(arq_velho)) or (not PathValido(arq_novo))) or
         (cmpstr(arq_velho, arq_novo))
priority ALTA
begin
   if cmpstr(arq_velho, arq_novo) then
      output C.confirm_gfs_rename(arq_velho, NOMES_ARQ_IGUAIS, false)
   else
```

```
      output C.confirm_gfs_rename(arq_velho, PATH_INVALIDO, false)
end;

when C.gfs_write(fid, buf, count)  { usuario solicita escrever no arquivo }
provided fid >= 0
priority ALTA
begin
   indice := VerificaArquivoTab(string_vazia, fid);
   { arquivo esta' aberto ou esta' sendo fechado }
   if indice <> IndiceInvalido then
      case TabArquivos[indice].acesso of
         ABERTO: begin
                    TabArquivos[indice].acesso := ESCREVENDO;
                    GeraNroSeq(indice, k);
                    { o arquivo esta' aberto }
                    output MA[indice].gfs_write(fid, buf, count);
                 end;
         { o arquivo ja' esta' sendo escrito -> ignora mensagem }
         ESCREVENDO: begin  end;
         { outra operacao esta' sendo realizada sobre o arquivo }
         ABRINDO, FECHANDO, LENDO, RENOMEANDO, REMOVENDO:
                              output C.respwrite(fid, 0, ERRO_OPERACAO_ILEGAL)
      end
   else
      output C.respwrite(fid, 0, ERRO_OPERACAO_ILEGAL) { o arquivo nao existe }
end;

when C.gfs_write(fid, buf, count)
provided fid < 0
priority ALTA
begin
   output C.respwrite(fid, 0, FID_INVALIDO)
end;

when C.gfs_read(fid, buf, count)   { usuario solicita ler o arquivo }
provided fid >= 0
priority ALTA
begin
   indice := VerificaArquivoTab(string_vazia, fid);
   { arquivo esta' aberto ou esta' sendo fechado }
   if indice <> IndiceInvalido then
      case TabArquivos[indice].acesso of
         ABERTO: begin
                    TabArquivos[indice].acesso := LENDO;
                    GeraNroSeq(indice, k);
                    { o arquivo esta' aberto }
                    output MA[indice].gfs_read(fid, buf, count);
                 end;
         LENDO: begin { o arquivo ja' esta' sendo lido -> ignora mensagem } end;
         { outra operacao esta' sendo realizada no arquivo }
         ESCREVENDO, ABRINDO, FECHANDO, RENOMEANDO, REMOVENDO:
                          output C.respread(fid, buf, 0, ERRO_OPERACAO_ILEGAL)
      end
   else
      output C.respread(fid, buf, 0, ERRO_OPERACAO_ILEGAL){o arquivo nao existe}
end;

when C.gfs_read(fid, buf, count)
provided fid < 0
priority ALTA
begin
   output C.respread(fid, buf, 0, FID_INVALIDO)
end;

{ usuario solicita ajustar a posicao do arquivo }
when C.gfs_lseek(fid, offset, origem)
provided fid >= 0
priority ALTA
```

```
      begin
         indice := VerificaArquivoTab(string_vazia, fid);
         { arquivo esta' aberto ou esta' sendo fechado }
         if indice <> IndiceInvalido then
         begin
            if TabArquivos[indice].acesso = ABERTO then
               { o arquivo esta' aberto }
               output MA[indice].gfs_lseek(fid, offset, origem)
            else
               { outra operacao esta' sendo realizada no arquivo }
               output C.confirm_gfs_lseek(fid, ERRO_OPERACAO_ILEGAL)
         end
         else
            { o arquivo nao existe }
            output C.confirm_gfs_lseek(fid, ERRO_OPERACAO_ILEGAL)
      end;

      when C.gfs_lseek(fid, offset, origem)
      provided fid < 0
      priority ALTA
      begin
         output C.confirm_gfs_lseek(fid, FID_INVALIDO)
      end;

      when C.gfs_close(fid, espera)
      provided fid >= 0                          { verifica se o fid e' valido }
      priority ALTA
      begin
         indice := VerificaArquivoTab(string_vazia,fid);
         if indice <> IndiceInvalido then   { arquivo esta' aberto ou sendo fechado }
             case TabArquivos[indice].acesso of
                ABERTO: begin
                           output MA[indice].gfs_close(fid, espera);
                           GeraNroSeq(indice, k);
                           { indica na Tabela de Arquivos que o arquivo esta' sendo
                             fechado }
                           TabArquivos[indice].acesso := FECHANDO;
                        end;
                FECHANDO: begin { ignora pclose -> retransmissao } end;
                { alguma outra operacao esta' sendo realizada sobre o arquivo }
                LENDO, ESCREVENDO, RENOMEANDO, REMOVENDO, ABRINDO:
                               output C.confirm_gfs_close(-2)
             end
         else
            { arquivo nao existe ou esta' sendo renomeado ou removido }
            output C.confirm_gfs_close(-2)
      end;

      when C.gfs_close(fid, espera)
      provided fid < 0
      priority ALTA
      begin
         output C.confirm_gfs_close(fid)
      end;

{ comunicacao entre as instancias InstCli do Cliente e o Usuario ---------------- }

   trans
      any NCli: 1..Cli do
      when MA[NCli].confirm_gfs_open(fid)
      provided (fid >= 0)    { arquivo foi aberto com sucesso }
      priority MEDIA
         begin
            { insere fid do arquivo na Tabela do Cliente }
            InsereFidTab(fid, NCli, TabArquivos);
            TabArquivos[NCli].acesso := ABERTO;
            output C.confirm_gfs_open(fid)
         end;
```

```
   when MA[NCli].confirm_gfs_open(fid)
   provided  (fid < 0)  { Cliente nao conseguiu abrir arquivo }
   priority MEDIA
      begin
         { elimina a instancia InstCli para aquele arquivo }
         release Instancia_Cliente[NCli];
         { retira o arquivo da Tabela de Arquivos do Cliente }
         RetiraArquivoTab(NCli, TabArquivos);
         output C.confirm_gfs_open(fid)
      end;

   when MA[NCli].respread(fid, buffer, n_bytes, status)
   priority MEDIA
      begin
         output C.respread(fid, buffer, n_bytes, status);
         TabArquivos[NCli].acesso := ABERTO
      end;

   when MA[NCli].respwrite(fid, n_bytes, status)
   priority MEDIA
      begin
         output C.respwrite(fid, n_bytes, status);
         TabArquivos[NCli].acesso := ABERTO
      end;

   when MA[NCli].confirm_gfs_close(fid)
   priority MEDIA
      begin
         { retira o arquivo da Tabela de Arquivos do Cliente }
         RetiraArquivoTab(NCli, TabArquivos);
         { elimina a instancia InstCli para aquele arquivo }
         release Instancia_Cliente[NCli];
         output C.confirm_gfs_close(fid)
      end;

   when MA[NCli].confirm_gfs_lseek(fid, status)
   priority MEDIA
      begin
         output C.confirm_gfs_lseek(fid, status)
      end;

   when MA[NCli].confirm_gfs_unlink(arq, status)
   priority MEDIA
      begin
         { retira o arquivo da Tabela de Arquivos do Cliente }
         RetiraArquivoTab(NCli, TabArquivos);
         { elimina a instancia InstCli para aquele arquivo }
         release Instancia_Cliente[NCli];
         output C.confirm_gfs_unlink(arq, status)
      end;

   when MA[NCli].confirm_gfs_rename(arq, status, f_serv)
   priority MEDIA
      begin
         { retira o arquivo da Tabela de Arquivos do Cliente }
         RetiraArquivoTab(NCli, TabArquivos);
         { elimina a instancia InstCli para aquele arquivo }
         release Instancia_Cliente[NCli];
         output C.confirm_gfs_rename(arq, status, f_serv)
      end;

{ comunicacao entre as instancias e a Rede ------------------------------- }

trans
   any NCli: 1..Cli do
   when MB[NCli].gfs_sopen(fid, arq, modo, lista_serv, ind_inst, ind_cli, nro_seq)
   priority MEDIA
```

```
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.gfs_sopen(fid, arq, modo, lista_serv, ind_inst, ind_cli,
                      RecuperaNroSeq(NCli))
end;

when MB[NCli].gfs_sclose(fid, lista_serv, ind_inst, ind_cli, nro_seq)
priority MEDIA
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.gfs_sclose(fid, lista_serv, ind_inst, ind_cli, RecuperaNroSeq(NCli))
end;

when MB[NCli].gfs_sunlink(fid, lista_serv, ind_inst, ind_cli, nro_seq)
priority MEDIA
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.gfs_sunlink(fid, lista_serv, ind_inst, ind_cli,
                        RecuperaNroSeq(NCli))
end;

when MB[NCli].gfs_srename(fid, arq_velho, arq_novo, lista_serv, ind_inst,
                          ind_cli, nro_seq)
priority MEDIA
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.gfs_srename(fid, arq_velho, arq_novo, lista_serv, ind_inst, ind_cli,
                        RecuperaNroSeq(NCli))
end;

when MB[NCli].gfs_sread(fid, lista_serv, ind_cli, nro_seq)
priority MEDIA
begin
   output R.gfs_sread(fid, lista_serv, ind_cli, RecuperaNroSeq(NCli))
end;

when MB[NCli].gfs_swrite(fid, lista_serv, ind_cli, nro_seq)
priority MEDIA
begin
   output R.gfs_swrite(fid, lista_serv, ind_cli, RecuperaNroSeq(NCli))
end;

when MB[NCli].reg_file(arq, modo, tam_unid, lista_serv, ind_inst, ind_cli,
                       ind_serv, nro_seq)
priority MEDIA
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.reg_file(arq, modo, tam_unid, lista_serv, ind_inst, ind_cli,
                     ind_serv, RecuperaNroSeq(NCli))
end;

when MB[NCli].reg_file_status(fid, status, ind_inst, ind_cli, ind_serv, nro_seq)
priority MEDIA
begin
   if(nro_seq = 0) then { primeira tx }
      GeraNroSeq(NCli, k);
   output R.reg_file_status(fid, status, ind_inst, ind_cli, ind_serv,
                            RecuperaNroSeq(NCli))
end;

when MB[NCli].reg_file_status_name(arq, status, ind_inst, ind_cli, ind_serv,
                                   nro_seq)
priority MEDIA
```

```
      begin
         if(nro_seq = 0) then { primeira tx }
            GeraNroSeq(NCli, k);
         output R.reg_file_status_name(arq, status, ind_inst, ind_cli, ind_serv,
                                       RecuperaNroSeq(NCli))
      end;

      when MB[NCli].reg_file_status_ren(arq, lista_serv, tam_unid, ind_inst, ind_cli,
                                        ind_serv, nro_seq)
      priority MEDIA
      begin
         if(nro_seq = 0) then { primeira tx }
            GeraNroSeq(NCli, k);
         output R.reg_file_status_ren(arq, lista_serv, tam_unid, ind_inst, ind_cli,
                                      ind_serv, RecuperaNroSeq(NCli))
      end;


      when MB[NCli].reg_file_status_rename(arq, fid, status, ind_inst, ind_cli,
                                           ind_serv, nro_seq)
      priority MEDIA
      begin
         if(nro_seq = 0) then { primeira tx }
            GeraNroSeq(NCli, k);
         output R.reg_file_status_rename(arq, fid, status, ind_inst, ind_cli,
                                         ind_serv, RecuperaNroSeq(NCli))
      end;



{ comunicacao entre a Rede e as instancias  ---------------------------- }

   trans
      when R.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
      priority BAIXA
      begin
         { verifica se a instancia existe }
         if TabArquivos[ind_inst].acesso <> DESOCUPADO then
            begin
               if ComparaNroSeq(ind_inst, nro_seq) then
                  output MB[ind_inst].confirm_sopen(fid, ind_serv, status, ind_inst,
                                 ind_cli, nro_seq)   { envia mensagem para instancia }
               else
                  begin  { ignora: retransmissao } end
            end
      end;

      when R.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
      { sclose -> operacao sincrona ou assincrona }
      priority BAIXA
      begin
         { verifica se a instancia existe }
         indice := VerificaArquivoTab(string_vazia, fid);
         if indice <> IndiceInvalido then
            begin
               if ComparaNroSeq(indice, nro_seq) then
                  output MB[indice].confirm_sclose(fid, ind_serv, status, ind_inst,
                                                   ind_cli, nro_seq)
               else
                  begin { ignora: retransmissao } end
         end
      end;

      when R.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
      { usado para confirmar o p_unlinnk -> operacao sincrona ou assincrona }
      priority BAIXA
      begin
         { verifica se a instancia existe }
```

```
   indice := VerificaArquivoTab(string_vazia, fid);
   if indice <> IndiceInvalido then
      begin
         if ComparaNroSeq(indice, nro_seq) then
            output MB[indice].confirm(fid, ind_serv, status, ind_inst, ind_cli,
                                      nro_seq)
         else
            begin { ignora: retransmissao } end
   end
end;

when R.confirmdado(fid, ind_serv, dados, status, nro_seq)
priority BAIXA
begin
   { verifica se a instancia existe }
   indice := VerificaArquivoTab(string_vazia, fid);
   if indice <> IndiceInvalido then
      begin
         if ComparaNroSeq(indice, nro_seq) then
            output MB[indice].confirmdado(fid, ind_serv, dados, status, nro_seq)
         else
            begin { ignora: retrasmissao } end
   end
end;

when R.confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
priority BAIXA
begin
   { verifica se a instancia existe }
   indice := VerificaArquivoTab(string_vazia, fid);
   if indice <> IndiceInvalido then
      begin
         if ComparaNroSeq(indice, nro_seq) then
            output MB[indice].confirmescrita(fid, ind_serv, status, ind_cli,
                                             nro_seq)
         else
            begin {ignora: retransmissao } end
    end
end;

when R.confirm_reg_file(fid, tam_unid, servidores, status, ind_inst, ind_cli,
                        nro_seq)
priority BAIXA
begin
   if TabArquivos[ind_inst].acesso <> DESOCUPADO then
   { verifica se a instancia existe }
      begin
         if ComparaNroSeq(ind_inst, nro_seq) then
            { envia mensagem para instancia }
            output MB[ind_inst].confirm_reg_file(fid, tam_unid, servidores,
                                       status, ind_inst, ind_cli, nro_seq)
         else
            begin  { ignora: retransmissao } end
      end
end;

when R.confirm_reg_status(fid, status, ind_inst, ind_cli, lista_serv, nro_seq)
priority BAIXA
begin
   if TabArquivos[ind_inst].acesso <> DESOCUPADO then
   { verifica se a instancia existe }
      begin
         if ComparaNroSeq(ind_inst, nro_seq) then
            output MB[ind_inst].confirm_reg_status(fid, status, ind_inst,
                                             ind_cli, lista_serv, nro_seq)
         else
            begin  { ignora: retransmissao } end
      end
```

```
      end;

      when R.confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst, ind_cli,
                             nro_seq)
      priority BAIXA
      begin
         if TabArquivos[ind_inst].acesso <> DESOCUPADO then
         { verifica se a instancia existe }
            begin
               if ComparaNroSeq(ind_inst, nro_seq) then begin
                  { registra o fid recuperado do metaservidor }
                  TabArquivos[ind_inst].fid := fid;
                  output MB[ind_inst].confirm_reg_arq(fid, status, tam_unid,
                                       lista_serv, ind_inst, ind_cli, nro_seq)
               end
               else
                  begin  { ignora: retransmissao } end
            end
      end;

      when R.confirm_reg_na(fid, status, ind_inst, ind_cli, nro_seq)
      priority BAIXA
      begin
         if TabArquivos[ind_inst].acesso <> DESOCUPADO then
         { verifica se a instancia existe }
            begin
               if ComparaNroSeq(ind_inst, nro_seq) then
                  output MB[ind_inst].confirm_reg_na(fid, status, ind_inst, ind_cli,
                                                     nro_seq)
               else
                  begin  { ignora: retransmissao } end
            end
      end;


   end;  { fim da especificacao do Cliente }
{end;}

{ especificacao do Servidor  ============================================== }

module Servidor process (ind_serv:integer);
   ip S: ServidorRede(provedor);
end;

body CorpoServidor for Servidor;
   ip MsA: array [1..Serv] of ServidorRede(usuario);
   { definicao dos modulos filhos do Servidor }
   module InstServ process (ind_serv, ind_inst_serv:integer; var2_sim:tipo_var_sim);
   { ind_inst_serv e' utilizado apenas durante a simulacao }
      ip A: ServidorRede(provedor);
   end;

   body CorpoInstServ for Instserv;
   var status:TipoStatus;
       dados_lidos:TipoDado;
       { Fid com o qual a instancia manipula o arquivo localmente }
       fid_local:integer;

   state ESPERA, ABERTO;

   { definicao das funcoes de InstServ ------------------------------------ }

   function Abre_CriaSegmentosLocais(arq:TipoArquivo; modo:integer):integer;
   {primitive;}
   begin
      Abre_CriaSegmentosLocais := var2_sim.r_open
   end;
```

```
function RemoveSegmentosLocais(arq:TipoArquivo):integer;
{primitive;}
begin
   RemoveSegmentosLocais := var2_sim.r_unlink
end;

function RenomeiaSegmentosLocais(arq1,arq2:TipoArquivo):integer;
{primitive;}
begin
   RenomeiaSegmentosLocais := var2_sim.r_rename
end;

procedure LeBlocoSolicitado(fid:fd; var dados:TipoDado; var status:TipoStatus);
{primitive;}
begin
   dados := 10; { para valores inteiros }
   { retorno default para a simulacao -> operacao teve exito }
   status := var2_sim.r_read
end;

function EscreveBlocoRecebido(fid:fd; dados:TipoDado):TipoStatus;
{primitive;}
begin
   { retorno default para a simulacao -> operacao teve exito }
   EscreveBlocoRecebido := var2_sim.r_write
end;

function FechaSegmentosLocais(fid:fd):integer;
{primitive;}
begin
   FechaSegmentosLocais := var2_sim.r_close
end;


{ inicializacao }

initialize to ESPERA
begin
end;

{ transicoes de InstServ }

trans
   from ESPERA to ABERTO
   when A.gfs_open(fid, arq, modo, ind_inst, ind_cli, nro_seq)
  { Cliente solicita abertura de arquivo }
   begin
      fid_local := Abre_CriaSegmentosLocais(arq, modo);
      if fid_local >= 0 then
         { arquivo aberto com sucesso }
         output A.confirm_sopen(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
      else
         { erro ao abrir o arquivo }
         output A.confirm_sopen(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
   end;

   from ESPERA to same
   when A.gfs_unlink(fid, arq, ind_inst, ind_cli, nro_seq)
   { Cliente solicita apagar arquivo }
   begin
      if RemoveSegmentosLocais(arq) >= 0 then
         output A.confirm(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
      else
         output A.confirm(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
   end;

   from ESPERA to same
   when A.gfs_rename(fid, arq1, arq2, ind_inst, ind_cli, nro_seq)
```

```
  { Cliente solicita renomear arquivo }
   begin
      if RenomeiaSegmentosLocais(arq1, arq2) >= 0 then
         output A.confirm(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
      else
          output A.confirm(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
   end;

  from ABERTO to same
  when A.gfs_read(fid, ind_cli, nro_seq)
  { Cliente solicita a leitura de um bloco do arquivo }
  begin
     LeBlocoSolicitado(fid, dados_lidos, status);
     output A.confirm_dado(fid, ind_serv, ind_cli, dados_lidos, status, nro_seq)
  end;

  from ABERTO to same
  when A.gfs_write(fid, dados, ind_cli, nro_seq)
  { Cliente solicita escrever um bloco no arquivo }
  begin
     output A.confirm_escrita(fid, ind_serv, ind_cli,
                              EscreveBlocoRecebido(fid, dados), nro_seq)
  end;

  from ABERTO to same
  when A.gfs_open(fid, arq, modo, ind_inst, ind_cli, nro_seq)
  begin
     output A.confirm_sopen(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
  end;

 from ABERTO to ESPERA
 when A.gfs_close(fid, ind_inst, ind_cli, nro_seq)
 begin
    { Cliente solicita fechamento do arquivo }
    if FechaSegmentosLocais(fid_local) >= 0 then
       output A.confirm_sclose(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
    else
       output A.confirm_sclose(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
 end;

end; { CorpoInstServ}

{ Tipos e variaveis do Servidor }

const NRO_MSG = 100; { valor hipotetico -> o valor deve ser, pelo menos, igual ao
                       numero de arquivos que o Mestre pode controlar, para que,
                       no caso extremo, haja espaco suficiente para acomodar todas
                       as mensagens.  }

type
   TabelaS = array[1..Serv] of fd;
   i_m = record
         f:fd;
         arq:TipoArquivo;
         s:TipoStatus;
         i:integer;
         lista_serv:serv_list;
         t_un: integer;
         nro_seq:TipoNumeracaoMsg;
   end;

   meta_dados =  record    { metadados armazenados no Meta Servidor }
                 nome_arquivo:TipoArquivo;   { nome do arquivo }
                 fid:fd;                     { fid do arquivo }
                 { tamanho da unidade de distribuicao do arquivo }
                 tam_unid:TipoUnidade;
                 { condicao em que se encontra o arquivo }
                 status: TipoOperacao;
```

```
                { lista de servidores envolvidos com o arquivo }
                lista_servidores:serv_list;
                { lista de Clientes com o arquivo aberto }
                lista_clientes: cli_list
                end;
   TabM = array[1..N_Arquivos] of meta_dados;

var
   TabelaMetaDados: TabM;
   indice_f:integer;
   TabServidor:TabelaS;
   indice:integer;
   v_msg:array[1..NRO_MSG] of i_m;   { usado para retransmissao de mensagens }
   ind_msg:integer;
   status_aux:TipoStatus;
   status_aux_oper:TipoOperacao;
   retransmite: boolean;
   list_aux:serv_list; { lista de servidores }
   tam_aux: integer;
   fid_aux, fid_novo: fd;
   lista_cli_aux: cli_list;
   i: integer;
   n_cli: integer;
   var_sim: array[1..Serv] of tipo_var_sim;  { usado pela simulacao }
   string_vazia: TipoArquivo;

   { variaveis utilizadas durante a simulacao ---------------------------------- }

   ret_error_reg_file, ret_error_reg_file_status, ret_error_reg_file_status_name,
   ret_error_reg_file_status_name2, ret_error_reg_file_status_ren,
   ret_error_reg_file_status_rename, ret_error_reg_file_status_rename2,
   ret_error_reg_file_status_serv, ret_error_reg_file_status_serv2:boolean;

modvar InstanciaServ:array[1..Serv] of InstServ;

{ funcoes do Servidor ------------------------------------------------------ }

procedure ArmazenaInfo(f:fd; var TabS:TabelaS; var indice:integer);
{ armazena informacoes referentes a um arquivo e retorna o indice da Tabela do
  Servidor onde foram armazenadas }
var i:integer;
begin
   i := 1;
   while (i <= Serv) and (TabS[i] <> -1) do
      i := i+1;
   if i > Serv then
      { nao ha' espaco na Tabela do Servidor para um novo arquivo }
      indice := IndiceInvalido
   else  begin
      TabS[i] := f;
      indice := i;
   end
end; { ArmazenaInfo }

procedure IniciaTabServ(var TabS:TabelaS);
{ inicia a Tabela do Servidor }
var i:integer;
begin
   i := 1;
   while i <= Serv do begin
      TabS[i] := -1;     { fid invalido }
      i := i+1;
   end
end; { IniciaTabServ }

function ProcuraArquivo(f:fd):integer;
{ procura por um arquivo na Tabela do Servidor retornando seu indice }
var i:integer;
```

```
   begin
      i := 1;
      while (i <= Serv) and (TabServidor[i] <> f) do
         i := i+1;
      if i > Serv then
         { nao foi encontrado nenhum arquivo com o <fid> correspondente }
         ProcuraArquivo := IndiceInvalido
      else
         ProcuraArquivo := i
   end; { ProcuraArquivo }

   procedure RetiraArqTabServ(indice:integer; var TabS:TabelaS);
   { retira um arquivo da Tabela do Servidor }
   begin
      TabS[indice] := -1
   end; { RetiraArqTabServ }

   procedure DetServidores(var lista_servidores:serv_list);
   { determina quais servidores estao envolvidos com um determinado arquivo }
   { primitive;} { procedimento a cargo do implementador }
   begin
      { a lista de servidores pode ser manipulada pelo Edb durante a simulacao na
transicao que invoca os procedimentos
        para a determinacao dos servidores envolvidos durante uma operacao de abertura de
arquivo.   }
   end;


   { Funcoes relacionadas com o controle de clientes com o mesmo arquivo aberto }

   procedure ConsultaClienteAberto(var TabelaM:TabM; indice, i_cli:integer;
                                   var status:TipoOperacao);
   { verifica se o cliente esta' lista de Clientes que estao com o mesmo arquivo
     aberto ou esperando confirmacao de abertura para o mesmo. Se sim, remove o
     Cliente da lista  }
   var i:integer;
   begin
      { procura pelo Cliente }
      while (TabelaM[indice].lista_clientes[i].ident_cli <> -1) and
            (TabelaM[indice].lista_clientes[i].ident_cli <> i_cli) do
         i := i+1;
      if (TabelaM[indice].lista_clientes[i].ident_cli <> -1)  then
      begin
      { Cliente foi encontrado na lista -> retira cliente e ajusta posicoes na lista }
         while TabelaM[indice].lista_clientes[i].ident_cli <> -1 do
         begin
            TabelaM[indice].lista_clientes[i].ident_inst :=
                                   TabelaM[indice].lista_clientes[i+1].ident_inst;
            TabelaM[indice].lista_clientes[i].ident_cli :=
                                   TabelaM[indice].lista_clientes[i+1].ident_cli;
            TabelaM[indice].lista_clientes[i].nro_seq :=
                                   TabelaM[indice].lista_clientes[i+1].nro_seq;
            i := i+1;
         end;
         TabelaM[indice].lista_clientes[i].ident_inst := -1;
         TabelaM[indice].lista_clientes[i].ident_cli := -1;
         TabelaM[indice].lista_clientes[i].nro_seq := -1;
         if (TabelaM[indice].lista_clientes[1].ident_cli = -1) then
         { ha' apenas um Cliente com o arquivo aberto }
            TabelaM[indice].status := FECHANDO;
         { caso contrario, preserva o status do arquivo como aberto }
         status := TabelaM[indice].status
      end
   end; { ConsultaClienteAberto }

   { Funcoes relacionadas com rx de msgs ---------------------------------- }

    function cmpstr(str1, str2:TipoArquivo):boolean;
```

```
 { verifica se duas strings sao iguais }
 var i:integer;
 begin
    i := 1;
    while (i <= MAXCOMPR) and (str1[i] = str2[i]) do
       i := i+1;
    if i > MAXCOMPR then
       cmpstr := true
    else
       cmpstr := false
 end; { cmpstr }


procedure ConsultaNroSeq(var fid:fd; arq:TipoArquivo; ind_cli, nro_seq:integer;
                         var lista_s:serv_list; var status:TipoStatus;
                         var tam_unid:integer; var retransmite:boolean);
{ consulta e recupera mensagens para retransmissao }
var i,j:integer;
begin
   i := 1;
   if fid > 0 then begin
   { consulta e' feita baseando-se tambem no <fid> do arquivo }
      while (i <= ind_msg) and (not ((v_msg[i].f = fid) and
            (v_msg[i].i = ind_cli) and (v_msg[i].nro_seq = nro_seq))) do
         i := i+1;
   end
   else begin { consulta e' feita baseando-se tambem no nome do arquivo }
     while (i <= ind_msg) and (not ((cmpstr(v_msg[i].arq, arq)) and
           (v_msg[i].i = ind_cli) and (v_msg[i].nro_seq = nro_seq))) do
         i := i+1;
   end;
   if i <= ind_msg then
   { foi encontrado um registro com o mesmo <fid>, <ind_cli> e <nro_seq> }
   begin
      status := v_msg[i].s;
      tam_unid := v_msg[i].t_un;
      if fid < 0 then fid := v_msg[i].f;
      retransmite := true;
      j := 1;
      while v_msg[i].lista_serv[j] <> -1 do begin
        lista_s[j] := v_msg[i].lista_serv[j];
        j := j+1;
      end;
      lista_s[j] := -1; { fim do vetor de servidores }
   end
   else
      retransmite := false;
end; { ConsultaNroSeq }

procedure ArmazenaMsg(fid:fd; arq:TipoArquivo; nro_seq:integer;
            lista_serv:serv_list; status:TipoStatus; ind_cli, tam_unid:integer);
{ armazena a mensagem enviada para o Cliente }
var i,j:integer;
begin
   ind_msg := ind_msg+1;
   if ind_msg > NRO_MSG then     { retira a mensagem da lista e ajusta posicoes }
   begin
      i := 1;
      while i <= NRO_MSG do begin
         v_msg[i].f := v_msg[i+1].f;
         v_msg[i].arq := v_msg[i+1].arq;
         v_msg[i].i := v_msg[i+1].i;
         v_msg[i].s := v_msg[i+1].s;
         v_msg[i].nro_seq := v_msg[i+1].nro_seq;
         v_msg[i].t_un := v_msg[i+1].t_un;
         j := 1;
         while v_msg[i+1].lista_serv[j] <> -1 do begin
            v_msg[i].lista_serv[j] := v_msg[i+1].lista_serv[j];
```

```
            j := j+1;
         end;
         v_msg[i].lista_serv[j] := -1; { fim do vetor de servidores }
         i := i+1;
      end;
      ind_msg := NRO_MSG;    { insere mensagem na cauda da lista }
   end;
   v_msg[ind_msg].f := fid;
   v_msg[ind_msg].arq := arq;
   v_msg[ind_msg].i := ind_cli;
   v_msg[ind_msg].s := status;
   v_msg[ind_msg].nro_seq := nro_seq;
   v_msg[ind_msg].t_un := tam_unid;
   i := 1;
   while lista_serv[i] <> -1 do begin
      v_msg[ind_msg].lista_serv[i] := lista_serv[i];
      i := i+1;
   end;
   v_msg[ind_msg].lista_serv[i] := -1;  { fim do vetor de servidores }
end; { ArmazenaMsg }


{ simulacao ------------------------------------------------------------ }

procedure IniciaStatusServ;   { usado pela simulacao }
{ faz a inicializacao to tipo de status retornado pelas instancias do Servidor }
var i:integer;
begin
   for i:= 1 to Serv do begin
      with var_sim[i] do begin
         r_open := 0;
         r_close := 0;
         r_read := OK;
         r_write := OK;
         r_unlink := 0;
         r_rename := 0;
     end
  end
end; { IniciaStatus Serv }

{ funcoes associadas com os metadados ------------------------------ }

procedure IniciaTabMetaDados(var TabelaM:TabM; var indice_final:integer);
{ inicia a Tabela de metadados com valores iniciais }
var fid_inicial, fid_final, i, j:integer;
begin
   { determina intervalo de valores de fid baseado no ind_serv }
   fid_inicial := (N_arquivos div N_servidores)*(ind_serv-1);
   if (ind_serv <> N_Servidores) then
       indice_final := N_arquivos div N_servidores
   else  { ultimo servidor }
      indice_final := (N_arquivos div N_servidores) +
                      (N_arquivos mod N_servidores);
   for i := 1 to indice_final do begin
      TabelaM[i].status := DESOCUPADO;   { lugar vago na Tabela do Mestre }
      TabelaM[i].fid := fid_inicial;
      fid_inicial := fid_inicial+1;
      for j:=1 to MAXCOMPR do
         TabelaM[i].nome_arquivo[j] := ' ';
      TabelaM[i].lista_servidores[1] := -1;   { inicia lista de servidores }
      TabelaM[i].lista_clientes[1].ident_cli := -1;   { inicia lista de clientes }
   end;
end; { IniciaTabMetaDados }

procedure RegistraArquivo(var TabelaM:TabM; var fid:fd; arq:TipoArquivo;
  var tam:TipoUnidade;  var lista_s: serv_list; var status_reg: TipoOperacao;
  indice_final, i_inst, i_cli: integer; n_seq:TipoNumeracaoMsg; var n_cli:integer);
{ armazena o arquivo na Tabela de Metadados e devolve o indice onde o arquivo foi
```

```
  armazenado, caso haja espaco na Tabela de Meta Dados }
var i,j:integer;
begin
   i := 1;
   while (i <= indice_final) and (TabelaM[i].status <> DESOCUPADO) and
         (not cmpstr(arq, TabelaM[i].nome_arquivo)) do
      i := i+1;
   { O nome do arquivo a ser aberto ja se encontra na Tabela -> inclui cliente na
     lista de clientes com o arquivo aberto ou aguardando confirmacao dessa
     operacao }
   if cmpstr(arq, TabelaM[i].nome_arquivo) then begin
      j := 1;
      while (TabelaM[i].lista_clientes[j].ident_cli <> -1) and
            (j < Compr_lista_cli) and
            (TabelaM[i].lista_clientes[j].ident_cli <> i_cli) do
         j := j+1;
      if (j = Compr_lista_cli) then begin
      { nao ha espaco para um novo Cliente na Tabela }
         fid := -2;   { sem espaco para mais um Cliente }
         n_cli := Compr_lista_cli-1;
      end
      else
         if (TabelaM[i].lista_clientes[j].ident_cli = i_cli) then
         begin { Cliente ja esta cadastrado -> houve rx de pedido )
            fid := -3;  { Cliente ja cadastrado }
            n_cli := j;
         end
         else begin
            { armazena Cliente }
            TabelaM[i].lista_clientes[j].ident_cli := i_cli;
            TabelaM[i].lista_clientes[j].ident_inst := i_inst;
            TabelaM[i].lista_clientes[j].nro_seq := n_seq;
            TabelaM[i].lista_clientes[j+1].ident_cli := -1;
            { recupera lista de Servidores para o Cliente sendo armazenado }
            j := 1;
            while (TabelaM[i].lista_servidores[j] <> -1) do begin
               lista_s[j] :=  TabelaM[i].lista_servidores[j];
               j := j+1;
            end;
            { Recupera o tamanho da unidade de distribuicao de um arquivo que ja
              existe }
            tam := TabelaM[i].tam_unid;
            fid := TabelaM[i].fid;  { Retorna o fid do arquivo compartilhado }
            n_cli := j-1;
         end;
   end
   else begin
      if i > indice_final then begin  { nao foi encontrado espaco vazio na Tabela }
         fid := IndiceInvalido; { retorna indice invalido }
         n_cli := 0;
      end
      else begin
         with TabelaM[i] do begin
            nome_arquivo := arq;
            tam_unid := tam;
            status := status_reg;
            lista_clientes[1].ident_cli := i_cli;
            lista_clientes[1].ident_inst := i_inst;
            lista_clientes[1].nro_seq := n_seq;
            lista_clientes[2].ident_cli := -1;  { delimita lista de clientes }
            if lista_s[1] = -1 then  { verifica se a lista de servidores e' vazia }
               { MetaServidor determina os servidores envolvidos com o arquivo }
               DetServidores(lista_s);
            j := 1;
            while lista_s[j] <> -1 do begin
               lista_servidores[j] := lista_s[j];
               j := j+1;
            end;
```

```
               lista_servidores[j] := lista_s[j];
            n_cli := 1;
         end;
         fid := TabelaM[i].fid;
      end;
   end;
   status_reg := TabelaM[i].status;  { retorna o status atual do arquivo }
end; { RegistraArquivo }

procedure AlteraStatusMetaDado(var TabelaM:TabM; indice:integer;
                               status: TipoOperacao);
begin
   TabelaM[indice].status := status
end; { AlteraStatusMetaDado }


procedure AjustaFid(var TabelaM:TabM; fid:fd; var fid_novo:fd; indice:integer);
begin
   fid_novo := TabelaM[indice].fid;
   TabelaM[indice].fid := fid;
end; { AjustaFid }

function ConsultaArquivo(TabelaM:TabM; arq:TipoArquivo; fid:fd;
                         indice_final:integer):integer;
{ verifica se existe na Tabela de Metadados um arquivo com o nome <arq> ou o fid
  <fid>, retornando seu indice }
var i:integer;
begin
   i := 1;
   if fid = -1 then             { pesquisa e' feita pelo nome do arquivo }
   begin
      while (i <= indice_final) and (not cmpstr(TabelaM[i].nome_arquivo, arq)) do
         i := i+1;
      if i <= indice_final then { o arquivo foi encontrado }
         ConsultaArquivo := i
      else
         ConsultaArquivo := IndiceInvalido { o arquivo nao foi encontrado }
   end
   else begin                   { pesquisa e' feita pelo fid do arquivo }
      while (i <= indice_final) and (TabelaM[i].fid <> fid) do
         i := i+1;
      if (i <= indice_final) and (TabelaM[i].status <> DESOCUPADO) then
         { o arquivo foi encontrado }
         ConsultaArquivo := i
      else
         ConsultaArquivo := IndiceInvalido { o arquivo nao foi encontrado }
  end
end; { Consulta Arquivo }

procedure RecuperaMetaDados(TabelaM:TabM; indice:integer; var tam_aux:integer;
                            var list_aux:serv_list; indice_final:integer);
var i:integer;
begin
   with TabelaM[i] do begin
      tam_aux := tam_unid;
      i := 1;
      while lista_servidores[i] <> -1 do begin
         list_aux[i] := lista_servidores[i];
         i := i+1;
      end;
         list_aux[i] := lista_servidores[i];
  end;
end; { RecuperaMetaDados }

procedure RecuperaClientes(TabelaM:TabM; indice:integer;
                           var lista_cli_aux:cli_list);
{ Recupera a lista de Clientes associados a um mesmo arquivo distribuido }
var i:integer;
```

```
 begin
    i := 1;
    while TabelaM[indice].lista_clientes[i].ident_cli <> -1 do begin
       lista_cli_aux[i] := TabelaM[indice].lista_clientes[i];
       i := i+1;
    end;
    lista_cli_aux[i].ident_cli := -1;  { delimita a lista }
end; { RecuperaClientes }


 procedure RecuperaServidores(TabelaM:TabM; indice:integer;
                              var lista_serv:serv_list);
 { Recupera a lista de Clientes associados a um mesmo arquivo distribuido }
 var i:integer;
 begin
    i := 1;
    while TabelaM[indice].lista_servidores[i] <> -1 do begin
       lista_serv[i] := TabelaM[indice].lista_servidores[i];
       i := i+1;
    end;
    lista_serv[i] := -1;  { delimita a lista }
 end; { RecuperaServidores }


 { ------------------------------------------------------------------------ }


 state CONTROLANDO;
 initialize to CONTROLANDO
 begin
    IniciaTabMetaDados(TabelaMetaDados, indice_f);   { inicia Tabela de MetaDados }
    { inicia Tabela para armazenar info sobre os arquivos armazenados localmente }
    IniciaTabServ(TabServidor);
    ind_msg := 0;
    string_vazia[1] := ' ';

    { usado pela simulacao ------------------------------------------ }
    IniciaStatusServ;
    ret_error_reg_file := false;
    ret_error_reg_file_status := false;
    ret_error_reg_file_status_name := false;
    ret_error_reg_file_status_name2 := false; {usado para o caso de a mesma operacao
          fazer duas solicitacoes na mesma operacao, uma no inicio e uma no final da
          operacao }
    ret_error_reg_file_status_ren := false;
    ret_error_reg_file_status_rename := false;
    ret_error_reg_file_status_rename2 := false;
    ret_error_reg_file_status_serv := false;
    ret_error_reg_file_status_serv2 := false;
    { ---------------------------------------------------------------- }

 end;

 { mensagens dos modulos-filho do Servidor para o Modulo de Rede }
 trans
    from CONTROLANDO to same
    { solicitacao de registro de abertura de arquivo }
    when S.reg_file(arq, modo, tam_unid, lista_serv, ind_inst, ind_cli, ind_serv,
                    nro_seq)
    begin
       { Cliente solicitou registro de operacao de abertura de arquivo }
       lista_cli_aux[1].ident_inst := ind_inst;
       lista_cli_aux[1].ident_cli := ind_cli;
       lista_cli_aux[1].nro_seq := nro_seq;
       lista_cli_aux[2].ident_cli := -1;
       fid_aux := -1;
       ConsultaNroSeq(fid_aux, arq, ind_cli, nro_seq, list_aux, status_aux, tam_aux,
                      retransmite);
```

```
   if retransmite then
      output S.confirm_reg_file_serv(fid_aux, tam_aux, list_aux, status_aux,
                                     lista_cli_aux)
   else begin
      if ret_error_reg_file then   { utilizado na simulacao }
         output S.confirm_reg_file_serv(fid_aux, tam_unid, list_aux, ERRO,
                                        lista_cli_aux)
      else begin
         { Tenta registrar metadados do arquivo. Se um Cliente ja estiver com o
           arquivo aberto ou abrindo, o novo Cliente sera inserido na lista de
           clientes relacionados com o arquivo }
         { operacao sendo realizada sobre o Meta Servidor }
         status_aux_oper := CRIANDO;
         RegistraArquivo(TabelaMetaDados, fid_aux, arq, tam_unid, lista_serv,
                 status_aux_oper, indice_f, ind_inst, ind_cli, nro_seq, n_cli);
         if (fid_aux = -2) then
            { Nao ha espaco na Tabela para um novo arquivo distribuido }
            output S.confirm_reg_file_serv(fid_aux, tam_unid, list_aux, ERRO,
                                           lista_cli_aux)
         else
            if status_aux_oper = ABERTO  then
              { O arquivo ja existe e esta aberto. Alerta imediatamente o
                Cliente ue solicitou a abertura }
               output S.confirm_reg_file_serv(fid_aux, tam_unid, lista_serv,
                                              ARQ_ABERTO, lista_cli_aux)
               { lista_cli_aux encapsula os identificadores do Cliente e da
                 instancia e o nro_seq da msg de solicitacao de registro de
                 operacao de cada cliente }
            else
               if (n_cli < 2) then
                  { fid <> -2 ou fid >= 0 -> significa que metadados nao foram
                    armazenados ou que o cliente devera entrar em contato com os
                    servidores }
                  output S.confirm_reg_file_serv(fid_aux, tam_unid, lista_serv,
                                                 ARQ_ABRINDO, lista_cli_aux)
      end;
    end;
end;

from CONTROLANDO to same
{ mensagens do Modulo de Rede para os modulos-filho do Servidor }
when S.gfs_open(fid, arq, modo, ind_inst, ind_cli, nro_seq)
begin
   ConsultaNroSeq(fid, string_vazia, ind_cli, nro_seq, list_aux, status_aux,
                  tam_aux, retransmite);
   if retransmite then
      output S.confirm_sopen(fid, ind_serv, status_aux, ind_inst, ind_cli,
                             nro_seq)
   else begin
      indice := ProcuraArquivo(fid);
      if indice <> IndiceInvalido then{ verifica se o arquivo ja' esta' aberto }
         output MsA[indice].gfs_open(fid, arq, modo, ind_inst, ind_cli, nro_seq)
      else begin
         ArmazenaInfo(fid, TabServidor, indice);
         { o numero de instancias do Servidor ja' alcancou seu limite }
         if indice = IndiceInvalido then
            { retorna ERRO ao Cliente sinalizando impossibilidade de abrir o
              arquivo }
            output S.confirm_sopen(fid, ind_serv, ERRO, ind_inst, ind_cli,
                                   nro_seq)
         else begin
            { a variavel <var_sim> e' somente usada para simulacao }
            init InstanciaServ[indice] with CorpoInstServ(ind_serv,indice,
               var_sim[indice]);
            connect MsA[indice] to InstanciaServ[indice].A;
            output MsA[indice].gfs_open(fid, arq, modo, ind_inst, ind_cli,
                                        nro_seq)
         end
```

```
       end
    end
end;


from CONTROLANDO to same
when S.reg_file_status_serv(fid, status, ind_inst, ind_cli, nro_seq)
begin
   { Cliente confirma finalizacao de operacao (abertura, fechamento,...) ao Meta
     Servidor }
   ConsultaNroSeq(fid, string_vazia, ind_cli, nro_seq, list_aux, status_aux,
                  tam_aux, retransmite);
   lista_cli_aux[1].ident_inst := ind_inst;
   lista_cli_aux[1].ident_cli := ind_cli;
   lista_cli_aux[1].nro_seq := nro_seq;
   lista_cli_aux[2].ident_cli := -1;
   tam_aux := 0;
   if retransmite then
      output S.confirm_reg_status_serv(fid, status_aux, list_aux, lista_cli_aux)
   else begin
      if (status <> ARQ_FECHADO) and (ret_error_reg_file_status_serv) then
         output S.confirm_reg_status_serv(fid, ERRO, list_aux, lista_cli_aux)
      else
         if (status = ARQ_FECHADO) and (ret_error_reg_file_status_serv2) then
            output S.confirm_reg_status_serv(fid, ERRO, list_aux, lista_cli_aux)
         else begin
            indice := ConsultaArquivo(TabelaMetaDados, string_vazia, fid,
                                      indice_f);
            if indice = IndiceInvalido then  { O arquivo nao consta na Tabela de
                                               MetaDados }
               output S.confirm_reg_status_serv(fid, ERRO, list_aux,
                                                lista_cli_aux)
            else
            case status of
               ARQ_FECHANDO: begin
               { Cliente esta' solicitando fechamento de arquivo }
               { verifica se ha' mais de um Cliente com o mesmo arquivo aberto.
                 Se houver, remove Cliente e responde diretamente a ele }
                  ConsultaClienteAberto(TabelaMetaDados, indice, ind_cli,
                                        status_aux_oper); { se apenas um Cliente
                        mantem o arq aberto, muda o status deste para FECHANDO }
                  ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, ARQ_FECHADO,
                              ind_cli, tam_aux);
                               if (status_aux_oper = ABERTO) then
                                begin
                                { arquivo continua aberto -> mais de um Cliente
                                  registrado }
                                   output S.confirm_reg_status_serv(fid,
                                        ARQ_FECHADO, list_aux, lista_cli_aux);
                                        { informa ao Cliente que o arquivo
                                          ja'pode ser considerado fechado }
                                  tam_aux := 0;  { variavel auxiliar }
                               end
                               else begin
                                  AlteraStatusMetaDado(TabelaMetaDados, indice,
                                  FECHANDO);  { altera o status do arquivo }
                                  output S.confirm_reg_status_serv(fid,
                                         ARQ_FECHANDO, list_aux, lista_cli_aux);
                                         { O Cliente tera que contactar os
                                           Servidores para fechar o arquivo }
                               end;
                            end;
                ARQ_ABERTO: begin
                   { informar a todos os Clientes associados a um arquivo que o
                     mesmo foi aberto }
                    AlteraStatusMetaDado(TabelaMetaDados, indice, ABERTO);
                    RecuperaClientes(TabelaMetaDados, indice, lista_cli_aux);
                    { recupera lista de servidores envolvidos com um arquivo }
```

```
                    RecuperaServidores(TabelaMetaDados, indice, list_aux);
                   { para cada Cliente aguardando resultado da operacao, uma
                     msg para rx e armazenada }
                   i := 2;
                   while(lista_cli_aux[i].ident_cli <> -1) do begin
                      ArmazenaMsg(fid, string_vazia, lista_cli_aux[i].nro_seq,
                      list_aux, ARQ_ABERTO, lista_cli_aux[i].ident_cli,
                      tam_aux);
                      i := i+1;
                   end;
                   lista_cli_aux[1].nro_seq := nro_seq;
                   { Manter o mesmo nro_seq para todos os usuarios exceto para o
                     primeiro, que se incumbiu de realizar a operacao e teve o
                     seu nro_seq alterado }
                    ArmazenaMsg(fid, string_vazia, lista_cli_aux[1].nro_seq,
                     list_aux, ARQ_ABERTO, lista_cli_aux[1].ident_cli, tam_aux);
                    output S.confirm_reg_status_serv(fid, ARQ_ABERTO, list_aux,
                                                     lista_cli_aux);
                            end;
                ARQ_FECHADO: begin
                   AlteraStatusMetaDado(TabelaMetaDados, indice, FECHADO);
                   ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, ARQ_FECHADO,
                               lista_cli_aux[i].ident_cli, tam_aux);
                   output S.confirm_reg_status_serv(fid, ARQ_FECHADO, list_aux,
                                                    lista_cli_aux);
                            end;
            end; { case }
         end;
      end;
   end;

from CONTROLANDO to same
when S.gfs_close(fid, ind_inst, ind_cli, nro_seq)
begin
   ConsultaNroSeq(fid, string_vazia, ind_cli, nro_seq, list_aux, status_aux,
                  tam_aux, retransmite);
    if retransmite then   { verifica se a mensagem e' uma retransmissao }
       output S.confirm_sclose(fid, ind_serv, status_aux, ind_inst, ind_cli,
                               nro_seq)
    else begin
       indice := ProcuraArquivo(fid);
       if indice <> IndiceInvalido then    { o arquivo esta' aberto }
          { entra em contato com a instancia }
          output MsA[indice].gfs_close(fid, ind_inst, ind_cli, nro_seq)
       else
           output S.confirm_sclose(fid, ind_serv, OK, ind_inst, ind_cli, nro_seq)
    end
end;

when S.gfs_read(fid, ind_cli, nro_seq)
begin
   output MsA[ProcuraArquivo(fid)].gfs_read(fid, ind_cli, nro_seq)
end;

when S.gfs_write(fid, dados, ind_cli, nro_seq)
begin
   output MsA[ProcuraArquivo(fid)].gfs_write(fid, dados, ind_cli, nro_seq)
end;

from CONTROLANDO to same
when S.reg_file_status_name(arq, status, ind_inst, ind_cli, ind_serv, nro_seq)
begin
   { Cliente solicita registro da situacao de arquivo ao Meta Servidor }
   fid_aux := -1;
   ConsultaNroSeq(fid_aux, arq, ind_cli, nro_seq, list_aux, status_aux, tam_aux,
                  retransmite);
   if retransmite then
      output S.confirm_reg_arq(fid_aux, status_aux, tam_aux, list_aux, ind_inst,
```

```
                                ind_cli, nro_seq)
   else begin
      tam_aux := 0;
      list_aux[1] := -1;
      if (status <> ARQ_REMOVIDO) and (ret_error_reg_file_status_name) then
         { utilizado para a simulacao }
         output S.confirm_reg_arq(-1, ERRO, tam_aux, list_aux, ind_inst, ind_cli,
                                  nro_seq)
      else
         if (status = ARQ_REMOVIDO) and (ret_error_reg_file_status_name2) then
            output S.confirm_reg_arq(-1, ERRO, tam_aux, list_aux, ind_inst,
                                     ind_cli, nro_seq)
         else begin
            indice := ConsultaArquivo(TabelaMetaDados, arq, -1, indice_f);
            if( indice <> IndiceInvalido) then begin
               RecuperaMetaDados(TabelaMetaDados, indice, tam_aux, list_aux,
                                 indice_f);
               if status = ARQ_REMOVIDO then
                  AlteraStatusMetaDado(TabelaMetaDados, indice, DESOCUPADO)
               else
                  case status of
                     ARQ_REMOVENDO:  AlteraStatusMetaDado(TabelaMetaDados,
                                     indice, REMOVENDO);
                     ARQ_RENOMEANDO: AlteraStatusMetaDado(TabelaMetaDados,
                                     indice, RENOMEANDO);
                     ARQ_FECHADO:    AlteraStatusMetaDado(TabelaMetaDados,
                                     indice, FECHADO);
                  end;
               output S.confirm_reg_arq(TabelaMetaDados[indice].fid, OK, tam_aux,
                                        list_aux, ind_inst, ind_cli, nro_seq);
            end
            else
               output S.confirm_reg_arq(-1, ERRO, tam_aux, list_aux, ind_inst,
                                        ind_cli, nro_seq);
         end;
      end;
end;


from CONTROLANDO to same
when S.reg_file_status_rename(arq, fid, status, ind_inst, ind_cli, ind_serv,
                              nro_seq)
{ Metaservidor recebe pedido para registrar a renomeacao do arquivo }
begin
   ConsultaNroSeq(fid, arq, ind_cli, nro_seq, list_aux, status_aux, tam_aux,
                  retransmite);
   if retransmite then
      output S.confirm_reg_arq(fid, status_aux, tam_aux, list_aux, ind_inst,
                               ind_cli, nro_seq)
   else begin
      list_aux[1] := -1;
      if (status = RENOMEADO_M1) and (ret_error_reg_file_status_rename) then
         output S.confirm_reg_arq(fid, ERRO, 0, list_aux, ind_inst, ind_cli,
                                  nro_seq)
      else
         if (status = ARQ_RENOMEADO) and (ret_error_reg_file_status_rename2) then
            output S.confirm_reg_arq(fid, ERRO, 0, list_aux, ind_inst, ind_cli,
                                     nro_seq)
         else begin
            indice := ConsultaArquivo(TabelaMetaDados, arq, -1, indice_f);
            if (indice <> IndiceInvalido) then begin
               RecuperaMetaDados(TabelaMetaDados, indice, tam_aux, list_aux,
                                 indice_f);
               AjustaFid(TabelaMetaDados, fid, fid_novo, indice);
               if (status = RENOMEADO_M1) then begin
                  { remove arquivo da Tabela de Arquivos }
                  AlteraStatusMetadado(TabelaMetaDados, indice, DESOCUPADO);
                  fid_novo := fid;
```

```
               end
               else  { status = RENOMEADO_M2 }
                  { disponibiliza novamente arquivo ao sistema }
                  AlteraStatusMetadado(TabelaMetaDados, indice, FECHADO);
               output S.confirm_reg_arq(fid_novo, OK, tam_aux, list_aux,
                                        ind_inst, ind_cli, nro_seq);
            end
            else
               output S.confirm_reg_arq(fid, ERRO, 0, list_aux, ind_inst,
                                        ind_cli, nro_seq);
         end;
   end;
end;


 when S.gfs_unlink(fid, arq, ind_inst, ind_cli, nro_seq)
 begin
    ConsultaNroSeq(fid, string_vazia, ind_cli, nro_seq, list_aux, status_aux,
                   tam_aux, retransmite);
    if retransmite then
       output S.confirm(fid, ind_serv, status_aux, ind_inst, ind_cli, nro_seq)
    else begin
       indice := ProcuraArquivo(fid);
       if indice <> IndiceInvalido then
       { verifica se o arquivo ja' esta' sendo manipulado -> precaucao contra
         retransmissoes }
       begin end { ignorar mensagem (servico em andamento) }
       else begin
          ArmazenaInfo(fid, TabServidor, indice);
          if indice = IndiceInvalido then
          { o numero de instancias do Servidor ja' alcancou seu limite }
          { retorna ERRO ao Cliente sinalizando impossibilidade de remover o arq}
             output S.confirm(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
          else begin
             init InstanciaServ[indice] with CorpoInstServ(ind_serv,indice,
                     var_sim[indice]);  { a variavel <var_sim> e' somente usada
                                          para simulacao }
             connect MsA[indice] to InstanciaServ[indice].A;
             output MsA[indice].gfs_unlink(fid, arq, ind_inst, ind_cli, nro_seq)
          end
       end
    end
 end;

 when S.reg_file_status_ren(arq, lista_serv, tam_unid, ind_inst, ind_cli,
                            ind_serv, nro_seq)
 { Registrar operacao de renomeacao no segundo MetaServidor }
 begin
    fid_aux := -1;
    ConsultaNroSeq(fid_aux, arq, ind_cli, nro_seq, list_aux, status_aux, tam_aux,
                   retransmite);
    if retransmite then
       output S.confirm_reg_na(fid_aux, status_aux, ind_inst, ind_cli, nro_seq)
    else begin
       if ret_error_reg_file_status_ren then
          output S.confirm_reg_na(fid_aux, ERRO, ind_inst, ind_cli, nro_seq)
       else begin
          status_aux_oper := RENOMEANDO;
          RegistraArquivo(TabelaMetaDados, fid_aux, arq, tam_unid, lista_serv,
                  status_aux_oper, indice_f, ind_inst, ind_cli, nro_seq, n_cli);
          { fid_aux = -1 significa que matadados nao foram armazenados }
          if (fid_aux >= 0) then
             output S.confirm_reg_na(fid_aux, OK, ind_inst, ind_cli, nro_seq)
          else
             output S.confirm_reg_na(fid_aux, ERRO, ind_inst, ind_cli, nro_seq)
       end;
    end;
 end;
```

```
 when S.gfs_rename(fid, arq1, arq2, ind_inst, ind_cli, nro_seq)
 begin
    ConsultaNroSeq(fid, string_vazia, ind_cli, nro_seq, list_aux, status_aux,
                   tam_aux, retransmite);
    if retransmite then
       output S.confirm(fid, ind_serv, status_aux, ind_inst, ind_cli, nro_seq)
    else begin
       indice := ProcuraArquivo(fid);
       if indice <> IndiceInvalido then
       { verifica se o arquivo ja' esta' sendo manipulado -> precaucao contra
         retransmissoes }
       begin end { ignorar mensagem -> retransmissao }
       else begin
          ArmazenaInfo(fid, TabServidor, indice);
          if indice = IndiceInvalido then
         { o numero de instancias do Servidor ja' alcancou seu limite }
         { retorna ERRO ao Cliente sinalizando impossibilidade de renomear o arq}
             output S.confirm(fid, ind_serv, ERRO, ind_inst, ind_cli, nro_seq)
          else begin
             init InstanciaServ[indice] with CorpoInstServ(ind_serv,indice,
                                              var_sim[indice]); { a variavel
                                    <var_sim> e' somente usada para simulacao }
             connect MsA[indice] to InstanciaServ[indice].A;
             output MsA[indice].gfs_rename(fid, arq1, arq2, ind_inst, ind_cli,
                                           nro_seq)
          end
       end
    end
 end;

{ mensagens das instancias do Servidor para o Clinte ------------------- }

 trans
 from CONTROLANDO to same
 any NIServ:1..Serv do
    when MsA[NIServ].confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli,
                                   nro_seq)
    provided status = OK
    begin
       list_aux[1] := -1;
       ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, status, ind_cli,
                   tam_aux);
       output S.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
    end;

    when MsA[NIServ].confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli,
                                   nro_seq)
    provided status = ERRO
    { verifica se houve erro ao abrir o arquivo }
    begin
       list_aux[1] := -1;
       ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, status, ind_cli,
                   tam_aux);
       RetiraArqTabServ(NIServ, TabServidor);
       release InstanciaServ[NISERV];  { elimina instancia }
       output S.confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
    end;

    when MsA[NIServ].confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli,
                                    nro_seq)
    { elimina a instancia independente se o arquivo fechou com sucesso ou nao }
    begin
       ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, status, ind_cli,
              tam_aux); { armazena a resposta para o caso de sua retransmissao }
       RetiraArqTabServ(NIServ, TabServidor);
       release InstanciaServ[NIServ];  { elimina instancia }
       output S.confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
```

```
         end;

         when MsA[NIServ].confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
         begin    { usado para o gfs_unlink e o gfs_rename }
            ArmazenaMsg(fid, string_vazia, nro_seq, list_aux, status, ind_cli,
                    tam_aux); { armazena a resposta para o caso de sua retransmissao }
            RetiraArqTabServ(NIServ, TabServidor);
            release InstanciaServ[NIServ];  { elimina instancia }
            output S.confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
         end;

         when MsA[NIServ].confirm_dado(fid, ind_serv, ind_cli, dados, status, nro_seq)
         begin
            output S.confirm_dado(fid, ind_serv, ind_cli, dados, status, nro_seq)
         end;

         when MsA[NIServ].confirm_escrita(fid, ind_serv, ind_cli, status, nro_seq)
         begin
            output S.confirm_escrita(fid, ind_serv, ind_cli, status, nro_seq)
         end;

   end; { Corpo do Servidor }

{ especificacao do Modulo de Rede ================================== }

module Rede process;
ip RC:array[1..N_usuarios] of ClienteRede(provedor);
   RS:array[1..N_Servidores] of ServidorRede(usuario);
end;

body CorpoRede for Rede;
   type TipoMsgTrans = (CS, SC);  { usado para simulacao }
        TipoIntTrans = (OPEN, CLOSE, RENAME, UNLINK, READ, WRITE, REG_FILE,
                        REG_FILE_STATUS, REG_FILE_STATUS2, REG_FILE_STATUS_NAME,
                        REG_FILE_STATUS_NAME2, REG_FILE_STATUS_REN,
                        REG_FILE_STATUS_RENAME, REG_FILE_STATUS_RENAME2,
                        CONFIRM_SOPEN, CONFIRM_SCLOSE, CONFIRM, CONFIRM_DADO,
                        CONFIRM_ESCRITA, CONFIRM_REG_FILE, CONFIRM_REG_STATUS_SERV,
                        CONFIRM_REG_ARQ, CONFIRM_REG_NA);  { usado para simulacao }
        msg_trans_cs = record
                          sopen, sclose, sunlink, srename, read, write, reg_file,
                          reg_file_status, reg_file_status_name,
                          reg_file_status_name2, reg_file_status_ren,
                          reg_file_status_rename, reg_file_status_rename2,
                          reg_file_status2: integer;
                       end;
        msg_trans_sc = record
                          confirm_sopen, confirm_sclose, confirm, confirm_dado,
                          confirm_escrita, confirm_reg_file, confirm_reg_status,
                          confirm_reg_arq, confirm_reg_na: integer;
                       end;
   var i:integer;      { variavel auxiliar }
       string_vazia: TipoArquivo; { variavel auxiliar }
       lista_aux:serv_list; { lista auxiliar }
       ClienteServidor, ServidorCliente:boolean; { usado para simulacao }
       nro_msg_perdidas_cs: msg_trans_cs;
       nro_msg_perdidas_sc: msg_trans_sc;

   state TRABALHANDO;

   function DetTrans(trans_exec:TipoMsgTrans; int_trans:TipoIntTrans):boolean;
   { Determina se uma mensagem sera' transmitida ou nao. Funcao usada para simulacao }
   var transmite:boolean;
   begin
      DetTrans := true;
      case trans_exec of
         CS: begin
                if (ClienteServidor) then
```

```
               DetTrans := true
            else begin
               case int_trans of
                  OPEN: if (nro_msg_perdidas_cs.sopen > 0) then DetTrans := false;
                  CLOSE: if (nro_msg_perdidas_cs.sclose > 0)
                              then DetTrans := false;
                  RENAME: if (nro_msg_perdidas_cs.srename > 0)
                              then DetTrans := false;
                  UNLINK: if (nro_msg_perdidas_cs.sunlink > 0)
                              then DetTrans := false;
                  READ:   if (nro_msg_perdidas_cs.read > 0)
                              then DetTrans := false;
                  WRITE:  if (nro_msg_perdidas_cs.write > 0)
                              then DetTrans := false;
                  REG_FILE: if (nro_msg_perdidas_cs.reg_file > 0)
                              then DetTrans := false;
                  REG_FILE_STATUS: if (nro_msg_perdidas_cs.reg_file_status > 0)
                              then DetTrans := false;
                  REG_FILE_STATUS2: if (nro_msg_perdidas_cs.reg_file_status2 > 0)
                              then DetTrans := false;
                  REG_FILE_STATUS_NAME:
                              if (nro_msg_perdidas_cs.reg_file_status_name > 0)
                              then DetTrans := false;
                  REG_FILE_STATUS_NAME2:
                              if (nro_msg_perdidas_cs.reg_file_status_name2 > 0)
                                 then DetTrans := false;
                  REG_FILE_STATUS_REN: if (nro_msg_perdidas_cs.reg_file_status_ren
                                          > 0) then DetTrans := false;
                  REG_FILE_STATUS_RENAME:
                              if (nro_msg_perdidas_cs.reg_file_status_rename > 0)
                                 then DetTrans := false;
                  REG_FILE_STATUS_RENAME2:
                              if (nro_msg_perdidas_cs.reg_file_status_rename2 > 0)
                                 then DetTrans := false;
               end;
            end;
         end;
      SC: begin
            if (ServidorCliente) then
               DetTrans := true
            else begin
               case int_trans of
                  CONFIRM_SOPEN: if (nro_msg_perdidas_sc.confirm_sopen > 0)
                     then DetTrans := false;
                  CONFIRM_SCLOSE: if (nro_msg_perdidas_sc.confirm_sclose > 0)
                     then DetTrans := false;
                  CONFIRM_DADO: if (nro_msg_perdidas_sc.confirm_dado > 0)
                     then DetTrans := false;
                  CONFIRM_ESCRITA: if (nro_msg_perdidas_sc.confirm_escrita > 0)
                     then DetTrans := false;
                  CONFIRM_REG_FILE: if (nro_msg_perdidas_sc.confirm_reg_file > 0)
                     then DetTrans := false;
                  CONFIRM_REG_STATUS_SERV:
                     if (nro_msg_perdidas_sc.confirm_reg_status > 0)
                        then DetTrans := false;
                  CONFIRM_REG_ARQ: if (nro_msg_perdidas_sc.confirm_reg_arq > 0)
                     then DetTrans := false;
                  CONFIRM_REG_NA: if (nro_msg_perdidas_sc.confirm_reg_na > 0)
                     then DetTrans := false;
               end;
            end;
         end;
   end
end; { DetTrans }

procedure checkpoint;
primitive;
```

```
initialize to TRABALHANDO
begin
  { usado para simulacao }
   ClienteServidor := true;
   ServidorCliente := true;
   with nro_msg_perdidas_cs do begin     { usado para simulacao }
      sopen := 0;
      sclose := 0;
      sunlink := 0;
      srename := 0;
      read := 0;
      write := 0;
      reg_file := 0;
      reg_file_status := 0;
      { usado no caso de o registro de servico ser feito duas vezes na mesma
        operacao (inicio e fim) }
      reg_file_status2 := 0;
      reg_file_status_name := 0;
      { usado no caso de o registro de servico ser feito duas vezes na mesma
        operacao (inicio e fim) }
      reg_file_status_name2 := 0;
      reg_file_status_ren := 0;
      reg_file_status_rename := 0;
   end;
   with nro_msg_perdidas_sc do begin     { usado para simulacao }
      confirm_sopen := 0;
      confirm_sclose := 0;
      confirm := 0;
      confirm_dado := 0;
      confirm_escrita := 0;
      confirm_reg_file := 0;
      confirm_reg_status := 0;
      confirm_reg_arq := 0;
      confirm_reg_na := 0;
  end;
end;

{ comunicacao dos Clientes com os Servidores }
trans
   from TRABALHANDO to same
   any NCli:1..N_usuarios do

   { Cliente solicita aos servidores abertura de arquivo }
   when RC[NCli].gfs_sopen(fid, arq, modo, lista_serv, ind_inst, ind_cli, nro_seq)
   begin
      i := 1;
      while lista_serv[i].ident_serv <> -1 do
      begin
         if not lista_serv[i].confirmado then
            { se o servidor ainda nao respondeu ao pedido }
            if DetTrans(CS, OPEN) then
               output RS[lista_serv[i].ident_serv].gfs_open(fid, arq, modo,
                                                    ind_inst, ind_cli, nro_seq)
            else
                nro_msg_perdidas_cs.sopen := nro_msg_perdidas_cs.sopen-1;
         i := i+1;
      end
   end;

   { Cliente solicita aos servidores fechamento de arquivo }
   when RC[NCli].gfs_sclose(fid,  lista_serv, ind_inst, ind_cli, nro_seq)
   begin
      i := 1;
      while lista_serv[i].ident_serv <> -1 do
      begin
         if not lista_serv[i].confirmado then
            { se o servidor ainda nao respondeu ao pedido }
            if DetTrans(CS, CLOSE) then
```

```
            output RS[lista_serv[i].ident_serv].gfs_close(fid,  ind_inst,
                                                          ind_cli, nro_seq)
         else
             nro_msg_perdidas_cs.sclose := nro_msg_perdidas_cs.sclose-1;
      i := i+1;
   end
end;


{ Cliente solicita aos servidores remocao de arquivo }
when RC[NCli].gfs_sunlink(fid, lista_serv, ind_inst, ind_cli, nro_seq)
 begin
   i := 1;
   while lista_serv[i].ident_serv <> -1 do
   begin
      if not lista_serv[i].confirmado then
      { se o servidor ainda nao respondeu ao pedido }
         if DetTrans(CS, UNLINK) then
            output RS[lista_serv[i].ident_serv].gfs_unlink(fid, string_vazia,
                                                    ind_inst, ind_cli, nro_seq)
         else
             nro_msg_perdidas_cs.sunlink := nro_msg_perdidas_cs.sunlink-1;
      i := i+1;
   end
end;

{ Cliente solicita aos servidores renomeacao de arquivo }
when RC[NCli].gfs_srename(fid, arq_velho, arq_novo, lista_serv,  ind_inst,
                          ind_cli, nro_seq)
begin
   i := 1;
   while lista_serv[i].ident_serv <> -1 do
   begin
      if not lista_serv[i].confirmado then
         { se o servidor ainda nao respondeu ao pedido }
         if DetTrans(CS, RENAME) then
            output RS[lista_serv[i].ident_serv].gfs_rename(fid, arq_novo,
                                       arq_velho, ind_inst, ind_cli, nro_seq)
         else
             nro_msg_perdidas_cs.srename := nro_msg_perdidas_cs.srename-1;
      i := i+1;
   end
end;

{ Cliente solicita aos servidores leitura de arquivo }
when RC[NCli].gfs_sread(fid, lista_serv, ind_cli, nro_seq)
begin
   i := 1;
   while lista_serv[i].ident_serv <> -1 do
   begin
      if not lista_serv[i].confirmado then
         { apenas a mensagem aos servidores nao confirmados }
         if DetTrans(CS, READ) then
            output RS[lista_serv[i].ident_serv].gfs_read(fid, ind_cli, nro_seq)
         else
             nro_msg_perdidas_cs.read := nro_msg_perdidas_cs.read-1;
      i := i+1;
   end
end;

{ Cliente solicita aos servidores leitura de arquivo }
when RC[NCli].gfs_swrite(fid, lista_serv, ind_cli, nro_seq)
begin
   i := 1;
   while lista_serv[i].ident_serv <> -1 do
   begin
      if not lista_serv[i].confirmado then
         { apenas envia a mensagem aos servidores nao confirmados }
```

```
          if DetTrans(CS, WRITE) then
             output RS[lista_serv[i].ident_serv].gfs_write(fid,
                                          lista_serv[i].dados, ind_cli, nro_seq)
          else
              nro_msg_perdidas_cs.write := nro_msg_perdidas_cs.write-1;
       i := i+1;
    end
 end;

{   ============================ Controle ================================ }


when RC[NCli].reg_file(arq, modo, tam_unid, lista_serv, ind_inst, ind_cli,
                       ind_serv, nro_seq)
begin
   if DetTrans(CS, REG_FILE) then
      output RS[ind_serv].reg_file(arq, modo, tam_unid, lista_serv, ind_inst,
                                   ind_cli, ind_serv, nro_seq)
    else
        nro_msg_perdidas_cs.reg_file := nro_msg_perdidas_cs.reg_file-1
end;

when RC[NCli].reg_file_status(fid, status, ind_inst, ind_cli, ind_serv, nro_seq)
begin
   if (status <> ARQ_FECHADO) then
      if DetTrans(CS, REG_FILE_STATUS) then
         output RS[ind_serv].reg_file_status_serv(fid, status, ind_inst, ind_cli,
                                                  nro_seq)
       else
         nro_msg_perdidas_cs.reg_file_status :=
            nro_msg_perdidas_cs.reg_file_status-1
   else
      if DetTrans(CS, REG_FILE_STATUS2) then
         output RS[ind_serv].reg_file_status_serv(fid, status, ind_inst, ind_cli,
                                                  nro_seq)
       else
         nro_msg_perdidas_cs.reg_file_status2 :=
            nro_msg_perdidas_cs.reg_file_status2-1
end;

when RC[NCli].reg_file_status_name(arq, status, ind_inst, ind_cli, ind_serv,
                                   nro_seq)
begin
   if (status <> ARQ_REMOVIDO) then
      if DetTrans(CS, REG_FILE_STATUS_NAME) then
         output RS[ind_serv].reg_file_status_name(arq, status, ind_inst, ind_cli,
                                                  ind_serv, nro_seq)
      else
          nro_msg_perdidas_cs.reg_file_status_name :=
             nro_msg_perdidas_cs.reg_file_status_name-1
   else
      if DetTrans(CS, REG_FILE_STATUS_NAME2) then
         output RS[ind_serv].reg_file_status_name(arq, status, ind_inst, ind_cli,
                                                  ind_serv, nro_seq)
      else
          nro_msg_perdidas_cs.reg_file_status_name2 :=
             nro_msg_perdidas_cs.reg_file_status_name2-1
end;


when RC[NCli].reg_file_status_ren(arq, lista_serv, tam_unid, ind_inst, ind_cli,
                                  ind_serv, nro_seq)
begin
   if DetTrans(CS, REG_FILE_STATUS_REN) then
      output RS[ind_serv].reg_file_status_ren(arq, lista_serv, tam_unid,
                                          ind_inst, ind_cli, ind_serv, nro_seq)
    else
        nro_msg_perdidas_cs.reg_file_status_ren :=
```

```
            nro_msg_perdidas_cs.reg_file_status_ren-1
   end;

   when RC[NCli].reg_file_status_rename(arq, fid, status, ind_inst, ind_cli,
                                        ind_serv, nro_seq)
   begin
     if (status <> RENOMEADO_M1) then
         if DetTrans(CS, REG_FILE_STATUS_RENAME2) then
            output RS[ind_serv].reg_file_status_rename(arq, fid, status, ind_inst,
                                                       ind_cli, ind_serv, nro_seq)
         else
             nro_msg_perdidas_cs.reg_file_status_rename2 :=
                nro_msg_perdidas_cs.reg_file_status_rename2-1
      else
         if DetTrans(CS, REG_FILE_STATUS_RENAME) then
            output RS[ind_serv].reg_file_status_rename(arq, fid, status, ind_inst,
                                                       ind_cli, ind_serv, nro_seq)
         else
             nro_msg_perdidas_cs.reg_file_status_rename :=
                nro_msg_perdidas_cs.reg_file_status_rename-1
   end;

 { Comunicacao entre os Servidores e os Clientes ------------------------ }

trans
   from TRABALHANDO to same
   any NServ:1..N_servidores do

   { o Servidor confirma ao Cliente pedido de abertura de arquivo }
   when RS[NServ].confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
   begin
      if DetTrans(SC, CONFIRM_SOPEN) then
         output RC[ind_cli].confirm_sopen(fid, ind_serv, status, ind_inst, ind_cli,
                                          nro_seq)
       else
           nro_msg_perdidas_sc.confirm_sopen := nro_msg_perdidas_sc.confirm_sopen-1
   end;


   { o Servidor confirma ao Cliente pedido de fechamento de arquivo }
   when RS[NServ].confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
   begin
      if DetTrans(SC, CONFIRM_SCLOSE) then
         output RC[ind_cli].confirm_sclose(fid, ind_serv, status, ind_inst, ind_cli,
                                           nro_seq)
       else
           nro_msg_perdidas_sc.confirm_sclose:= nro_msg_perdidas_sc.confirm_sclose-1
   end;

   { o Servidor confirma ao Cliente pedido de remocao ou renomeacao de arquivo }
   when RS[NServ].confirm(fid, ind_serv, status, ind_inst, ind_cli, nro_seq)
   begin
      if DetTrans(SC, CONFIRM) then
         output RC[ind_cli].confirm(fid, ind_serv, status, ind_inst, ind_cli,
                                    nro_seq)
       else
           nro_msg_perdidas_sc.confirm := nro_msg_perdidas_sc.confirm-1
   end;

   { o Servidor confirma ao Cliente pedido de leitura de arquivo }
   when RS[NServ].confirm_dado(fid, ind_serv, ind_cli, dados, status, nro_seq)
   begin
      if DetTrans(SC, CONFIRM_DADO) then
         output RC[ind_cli].confirmdado(fid, ind_serv, dados, status, nro_seq)
       else
           nro_msg_perdidas_sc.confirm_dado := nro_msg_perdidas_sc.confirm_dado-1
   end;
```

```
{ o Servidor confirma ao Cliente pedido de escrita de arquivo }
when RS[NServ].confirm_escrita(fid, ind_serv, ind_cli, status, nro_seq)
begin
   if DetTrans(SC, CONFIRM_ESCRITA) then
      output RC[ind_cli].confirmescrita(fid, ind_serv, status, ind_cli, nro_seq)
    else
        nro_msg_perdidas_sc.confirm_escrita :=
           nro_msg_perdidas_sc.confirm_escrita-1
end;

{ o Meta-Servidor confirma ao Cliente pedido de registro de arquivo }
when RS[NServ].confirm_reg_file_serv(fid, tam_unid, servidores, status,
                                     lista_cli)
begin
   if DetTrans(SC, CONFIRM_REG_FILE) then begin
      i := 1;
      while (lista_cli[i].ident_cli <> -1) do begin
         output RC[lista_cli[i].ident_cli].confirm_reg_file(fid, tam_unid,
            servidores, status, lista_cli[i].ident_inst, lista_cli[i].ident_cli,
            lista_cli[i].nro_seq);
         i := i+1;
      end;
    end
    else
        nro_msg_perdidas_sc.confirm_reg_file :=
           nro_msg_perdidas_sc.confirm_reg_file-1
end;


{ o Meta-Servidor confirma ao Cliente pedido de registro de status arquivo }
when RS[NServ].confirm_reg_status_serv(fid, status, lista_serv, lista_cli)
begin
   if DetTrans(SC, CONFIRM_REG_STATUS_SERV) then begin
      i := 1;
      while (lista_cli[i].ident_cli <> -1) do begin
         output RC[lista_cli[i].ident_cli].confirm_reg_status(fid, status,
            lista_cli[i].ident_inst, lista_cli[i].ident_cli, lista_serv,
            lista_cli[i].nro_seq);
         i := i+1;
      end;
   end
    else
        nro_msg_perdidas_sc.confirm_reg_status :=
           nro_msg_perdidas_sc.confirm_reg_status-1
end;


{ o Meta-Servidor confirma ao Cliente pedido de registro de status arquivo }
when RS[NServ].confirm_reg_arq(fid, status, tam_unid, lista_serv, ind_inst,
                               ind_cli, nro_seq)
begin
   if DetTrans(SC, CONFIRM_REG_ARQ) then
      output RC[ind_cli].confirm_reg_arq(fid, status, tam_unid, lista_serv,
                                         ind_inst, ind_cli, nro_seq)
    else
        nro_msg_perdidas_sc.confirm_reg_arq :=
           nro_msg_perdidas_sc.confirm_reg_arq-1
end;

{ o Meta-Servidor confirma ao Cliente pedido de registro de status arquivo }
when RS[NServ].confirm_reg_na(fid, status, ind_inst, ind_cli, nro_seq)
begin
   if DetTrans(SC, CONFIRM_REG_NA) then
      output RC[ind_cli].confirm_reg_na(fid, status, ind_inst, ind_cli, nro_seq)
    else
      nro_msg_perdidas_sc.confirm_reg_na := nro_msg_perdidas_sc.confirm_reg_na-1
end;
```

```
   end; { Modulo de Rede }

   modvar MCliente:array[1..N_usuarios] of Cliente;
          MServidor:array[1..N_servidores] of Servidor;
          MRede:Rede;

   initialize      { criacao estatica das ocorrencias }
   begin
      init MRede with CorpoRede;
      all i:1..N_servidores do
      begin init MServidor[i] with CorpoServidor(i);
            connect MServidor[i].S to MRede.RS[i]
      end;
      all i:1..N_usuarios do
      begin init MCliente[i] with CorpoCliente(i);
            connect MCliente[i].R to MRede.RC[i];
            attach UC[i] to MCliente[i].C;
      end;

   end;
 end; { Fim do Gerente }

 modvar MUsuario:array[1..N_usuarios] of Usuario;
        MGerente:Gerente;
 initialize      { criacao das ocorrencias estaticas do Usuario }
 begin
    init MGerente with CorpoGerente;
    all i:1..N_usuarios do
    begin init MUsuario[i] with CorpoUsuario(i);
          connect MUsuario[i].U to MGerente.UC[i];
    end;
 end;
end. { Fim da declaracao do Sistema de Arquivos descentralizado }
```