

Universidade Federal da Paraíba – UFPB
Centro de Ciências e Tecnologia - CCT
Departamento de Sistemas e Computação - DSC
Coordenação de Pós-Graduação em Informática -
COPIN

Serviço CORBA de Diagnóstico de Falhas

(SDF)

por

Marcela Santana Guimarães Batalha

Campina Grande, Outubro de 2001

MARCELA SANTANA GUIMARÃES BATALHA

Serviço CORBA de Diagnóstico de Falhas (SDF)

**Dissertação de Mestrado submetida à
Coordenação do Curso de Pós-
Graduação em Informática da
Universidade Federal da Paraíba –
Campus II como requisito parcial para
a obtenção do grau de Mestre em
Informática.**

Orientador: **Raimundo José de Araújo Macêdo**

Área de Concentração: **Ciência da Computação**

Campina Grande, Outubro de 2001

Ficha Catalográfica

BATALHA, Marcela Santana Guimarães

B328S

Serviço *CORBA* de Diagnóstico de Falhas (SDF).

Dissertação (Mestrado) – UFPB/CCT/COPIN, Campina Grande, Outubro de 2001.

189p. Il.

Orientador: Raimundo José de Araújo Macêdo

1. Sistemas Distribuídos
2. Detecção
3. Diagnóstico
4. Gerenciamento.

CDU: 681.3:066D

CDU


004.75(043)

SERVIÇO CORBA DE DIAGNÓSTICO DE FALHAS (SDF)

MARCELA SANTANA GUIMARÃES BATALHA

DISSERTAÇÃO APROVADA EM 25.10.2001


PROF. RAIMUNDO JOSÉ DE ARAÚJO MACÊDO, Ph.D
Orientador


PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Examinador


PROF. ANTONIO MAURO BARBOSA DE OLIVEIRA, Dr.
Examinador

CAMPINA GRANDE – PB

Agradecimentos

As pessoas que trabalharam ou trabalham no *LaSiD*, fazendo deste laboratório um lugar agradável de estar. À Socorro, Eduardo, Fábio, Deise, Criston, George, Flávio, Aline, Sérgio, Ana, Beto (s), que me ajudaram, incentivaram ou simplesmente estiveram presentes.

Ao prof. Raimundo Macêdo, meu orientador. Obrigada pela confiança, estímulo e especialmente pela oportunidade de aprender trabalhando ao seu lado.

A Luiza pelo carinho e disposição em auxiliar em todos os momentos. À Nívea pela amizade, paciência, e por tantas e tantas correções e palavras na hora certa. Muito obrigada.

A Isabela, Manuela e Rebecca. Vocês são as irmãs que eu não tive, mas que pude escolher. A Andréa pela ajuda em um momento tão difícil. Nunca vou esquecer. À Edna pelos momentos de descontração, quando eu mais precisava. À Anne por ser uma amiga de todas as horas. Sua firmeza e integridade sempre me impressionaram.

A Sóstenes pelo incentivo. Sem o seu apoio eu não teria começado este projeto. Obrigada.

A minha família, pequena mas unida. Obrigada tia Balila, tio João, Zeca, primos, sobrinhos, por vocês serem tão especiais e amorosos.

Ao meu pai Jorge pelo reencontro.

Aos meus pais, Márcia e Waldemar, que, de longe, sempre estiveram muito presentes, incentivando meus estudos, me apoiando incondicionalmente. A orientação de vocês continuam comigo, onde quer que eu vá.

A minha avó Helena, que é a força, o suporte e, acima de tudo, meu exemplo. Obrigada por ter me ensinado tantas coisas.... Todos na família devem a senhora tudo que são.

E acima de tudo, agradeço a meu Deus Jeová. Dele tiro toda serenidade e coragem para continuar minha vida e meus projetos. Como diz o texto bíblico “*Para todas as coisas tenho força em virtude daquele que me confere poder*” (*Filipenses 4:13*).

Resumo

Em sistemas distribuídos, falhas de processos ou de canais de comunicação, se não forem devidamente tratadas, podem resultar na interrupção das aplicações e, portanto, dos serviços relacionados. Em geral, o tratamento de falhas, seja pelo mascaramento destas ou pela recuperação do estado do sistema, requer em primeiro lugar o diagnóstico dos componentes falhos, identificando-os e informando aos processos operacionais, de uma forma consistente, o diagnóstico estabelecido.

Neste trabalho, apresentamos a concepção e desenvolvimento de um serviço sobre CORBA para o Diagnóstico de Falhas em Sistemas Distribuídos (SDF) onde limites de tempo (*timeouts*) não podem ser usados como indicação precisa de falhas. O serviço SDF é distribuído, tolerante a falhas e foi concebido sobre os conceitos de gerenciamento, diagnóstico e detecção de falhas em sistemas assíncronos. Para gerar diagnósticos utilizamos *timeouts* adaptáveis, consultas ao sistema operacional e um protocolo de acordo em duas fases (uma variação do *two-phase commit*) que garante uma visão consistente do diagnóstico entre os vários processos gerenciados.

O Serviço de Diagnóstico inclui uma ferramenta visualizadora que apresenta o estado dos processos monitorados e o histórico das máquinas destes processos. Ele foi implementado e testado num ambiente JAVA/CORBA na rede de computadores do LaSiD/UFBA.

Abstract

In distributed systems, process or communication channel failures, if not properly treated, may result in the complete interruption of applications and, therefore, the related services. In general, the failure treatment, by masking faults or failure recovery, requires in the first place the diagnosis of the faulty components, by identifying and informing them to the operational processes, in a consistent manner.

This thesis presents the development of a Fault Tolerant Diagnosis Service based on CORBA (SDF) for distributed systems where timeouts can not be used for a precise indication of component failures or diagnosis. The service SDF is distributed, fault tolerant and integrates concepts of management, diagnosis and failure detection in asynchronous systems. In order to establish a diagnosis we use adaptive timeouts, operational system calls and a variation of the *two-phase commit protocol* that guarantees a diagnosis coherent view between the managed processes.

The Diagnosis Service includes a visualisation tool that presents monitored process states and history of the machines where they are running. It was implemented and tested over a JAVA/CORBA environment on the LaSiD/UFBA computer network.

Sumário

Listas de Figuras	XII
Listas de Tabelas	XVI
1. Introdução	2
1.1. Estrutura da Dissertação	4
2. Detecção de falhas, Diagnóstico de Sistema e Gerenciamento em Sistemas Distribuídos	7
2.1. Terminologia Adotada na Dissertação	8
2.2. Modelos de Falhas	8
2.3. Modelos de Sistemas	9
2.3.1. Modelo Síncrono	10
2.3.2. Modelo Puramente Assíncrono	11
2.3.3. Modelo Quasi-Síncrono	12
2.3.4. Modelo Assíncrono Temporizado	13
2.4. Associando Modelos de Sistema e Modelos de Falha	14
2.5. Detecção de Falhas nos Modelos de Sistemas	15
2.6. Diagnóstico de Sistemas	18
2.7. Gerenciamento	20
2.8. Síntese dos Conceitos Apresentados	21
3. Serviço de Diagnóstico de Falhas e sua Arquitetura	24

3.1. Estrutura Conceitual do Serviço de Diagnóstico de Falhas	25
3.2. Arquitetura do Serviço de Diagnóstico de Falhas e seus Componentes	26
3.3. Adoção da plataforma CORBA para comunicação	29
3.4. Monitoração de Aplicações Distribuídas pelo SDF	31
3.5. Tipos de Diagnósticos de Falhas	32
3.5.1. Processo não Responde Dentro dos Limites de Tempo de Comunicação Especificados	33
3.5.2. Processo não Está mais na Tabela do Sistema Operacional ou não Responde ao Módulo do SDF Local	35
3.5.3. Falha na Máquina que Contém Processos Monitorados.	37
3.5.4. Falha do Módulo SDF de uma Máquina que Tenha Processos Monitorados em Estado Suspeito.	40
3.6. Protocolo de Acordo Utilizado no SDF	42
3.6.1. Fases do Protocolo <i>Two-Phase Commit</i> do SDF – Retirada de Máquinas ou Módulos Falhos	43
3.6.2. Fases do protocolo <i>Two-Phase Commit</i> do SDF – eleição de um novo coordenador	45
3.6.3. Considerações Sobre o Protocolo	48
3.7. Visualizador	49
3.8. Serviço SDF Estendido por Questões de Escalabilidade	59
3.9. Resumo do Serviço Apresentado	62
4. Detalhes de Implementação do Serviço de Diagnóstico de Falhas	65
4.1. Classes que Compõem o Núcleo Básico do SDF	65
4.1.1. Classe <i>ServicoReconfiguracao</i>	68

4.1.2. Classe ServicoDFGerente	74
4.1.3. Classe ServicoDFAgenteConexao	77
4.1.4. Classe ServicoDFAgenteObjetos	79
4.1.5. Classe ListaProcessosCorretosAplicacao	80
4.1.6. Classe ListaProcessosFalhosAplicacao	83
4.2. Outras Classes que auxiliam as Classes do núcleo básico	85
4.2.1. Classe AnalisadorDeConexoes	85
4.2.2. Classe Pinger	85
4.2.3. Classe ListaExcluidosParticionamento	86
4.2.4. Classe ThreadSolicitacaoInvestigacaoProcessoSO	87
4.2.5. Classe ThreadMonitoriaServidorNomes	88
4.2.6. Classe ThreadAtualizaHistorico e Classe ThreadHistorico	88
4.2.7. Classe ThreadAtualizaNumeroReconfiguracao e Classe ThreadNumeroEstadoReconfiguracao	89
4.2.8. Classe ThreadSolicitaVotos e Classe ThreadVotos	90
4.2.9. Classe TemporizadorCoordenador	91
4.2.10. Classe ThreadRetiraProcessoGrupo, Classe ThreadSolicitaRetiradaProcessoGrupo e Classe ThreadRetiraProcessosMaquinaFalha	91
4.2.11. Classe ThreadVotosProximoCoord	93
4.2.12. Classe TemporizadorEleicaoCoordenador	93
4.2.13. Classe ThreadInsereObjetoParaGerenciamento	94
4.3. Interfaces de Monitoria e Tipos de Processos Monitorados	95

4.3.1. Classes que Atuam Como Processos Interface	97
4.3.1.1. Detalhes de Implementação da Classe ProcessoCorreto	98
4.3.1.2. Detalhes de Implementação da Classe ProcessoInterfaceCorreio	98
4.3.1.3. Detalhes de Implementação da Classe ProcessoInterfaceWeb	99
4.3.1.4. Detalhes de Implementação da Classe ProcessoInterfaceServico	99
4.4. Classes que Compõem o Visualizador do Serviço de Diagnóstico de Falhas	100
4.5. Diagramas de Seqüência	101
4.5.1. Processo não Responde Dentro dos Limites de Tempo de Comunicação Especificado	102
4.5.2. Processo não Está Mais na Tabela do Sistema Operacional ou não Responde ao Módulo do SDF Local	103
4.5.3. Falha do Módulo SDF Local de uma Máquina que Tenha Processos Monitorados em Estado Suspeito	106
4.5.4. Falha na Máquina que Contém Processos Monitorados	107
4.5.4.1. Retorno ao Grupo de uma Máquina Falha	108
4.5.5. Diagramas de Seqüência do Protocolo de Acordo do SDF	110
4.6. Referência aos Objetos SDF no Servidor de Nomes	112
4.7. Comunicação Entre os Objetos do SDF Sobre a Plataforma CORBA	114
4.8. Resumo dos Aspectos Principais Apresentados no Capítulo	119
5. Testes e Avaliação de Desempenho	121
5.1. Ambiente dos Experimentos	121
5.2. Definição das Métricas	122
5.3. Descrição dos Experimentos e Objetivos	122

5.4. Amostras Para Utilização nos Testes	123
5.5. Resultados Estatísticos dos Testes	128
5.5.1. Retirada de Processos por Falhas Individuais	129
5.5.2. Retirada de Processos por Falha da Máquina Onde Este Reside	131
5.6. Síntese dos Resultados Encontrados nos Testes	133
6. Especificações de Tolerância a Falhas CORBA (FT-CORBA).	138
6.1. Conceitos Básicos e Infraestrutura do Padrão FT-CORBA	139
6.2. Serviços de Gerenciamento da Especificação FT-CORBA	142
6.2.1. Gerenciamento de Replicação	143
6.2.2. Gerenciamento de Falhas	144
6.2.3. Gerenciamento de Recuperação e <i>Logging</i>	145
6.3. Componentes FT-CORBA e o SDF	146
6.3.1. Classe <code>ProcessoInterfaceFTCORBA</code>	147
6.4. Resumo dos Principais Itens Vistos Neste Capítulo	149
7. Trabalhos Correlatos	152
7.1. GroupPac	152
7.1.1. Comparação dos Serviços GroupPac com o SDF	155
7.2. DOORS	156
7.2.1. Comparação dos Serviços DOORS com o SDF	159
7.3. OGS (The Corba Object Group Service)	160
7.3.1. Comparação do OGS com o SDF	163

7.4. Ferramenta Piranha	163
7.4.1. Ferramenta Piranha e Visualizador SDF	164
7.5. Comparação Final do SDF com os Trabalhos Correlatos Descritos	165
8. Conclusão e Trabalhos Futuros	168
8.1. Trabalhos em andamento e propostas para trabalhos futuros	170
BIBLIOGRAFIA	175
ANEXOS	181

Lista de Figuras

Figura 3.1 - Estrutura conceitual do SDF.....	26
Figura 3.2 - Módulos SDF em máquinas da rede	27
Figura 3.3 - SDF e seus sub-módulos e componentes	28
Figura 3.4 - Inscrição do módulo SDF no Servidor de Nomes CORBA	30
Figura 3.5 - Diferentes tipos de processos e serviços monitorados pelo SDF.....	32
Figura 3.6 - Estouro de tempo de comunicação do Processo P2	34
Figura 3.7 - Solicitação de investigação sobre processo no sistema operacional.....	36
Figura 3.8 - Agente de objeto detecta falha do processo P3 e objeto reconfiguração inicia uma reconfiguração.	37
Figura 3.9 - O módulo SDF da máquina 1 não consegue contactar a máquina 2 e seus processos..	38
Figura 3.10 - Máquina sem acesso ao servidor de nomes e as ações decorrentes desta situação	40
Figura 3.11 - Detecção de falha do módulo	41
Figura 3.12 - 1º Fase do Protocolo de acordo para retirada dos processos de uma máquina falha...	44
Figura 3.13 - 2º Fase do Protocolo de acordo para retirada de processos de uma máquina falha ...	45
Figura 3.14 - 1º Fase do Protocolo de eleição do novo coordenador do grupo	47
Figura 3.15 - 2º Fase do Protocolo de eleição do novo coordenador do grupo	48
Figura 3.16 - Tela inicial do Visualizador.....	52
Figura 3.17 - Tela de <i>acompanhamento dos processos</i>	53
Figura 3.18 - Tela de <i>histórico das máquinas</i>	53
Figura 3.19 - Tela de <i>acompanhamento dos processos</i> -suspeita de falha nos processos 333 e 444.	54
Figura 3.20 - Tela de <i>acompanhamento dos processos</i> com os processos 333 e 444 falhos.	55
Figura 3.21 - Tela de <i>histórico das máquinas</i> com a retirada da máquina 200.17.144.73	55
Figura 3.22 - Tela de <i>histórico das máquinas</i> com o retorno da máquina 200.17.144.73	56
Figura 3.23 - Tela de <i>acompanhamento dos processos</i> com falha do processo 111.....	57
Figura 3.24 - Tela de <i>histórico das máquinas</i> - retirada da máquina 200.17.144.67 por falha no módulo	58
Figura 3.25 - Tela de <i>acompanhamento dos processos</i> - falha do módulo SDF da máquina 200.17.144.67.	58
Figura 3.26 - Monitoria dos processos que estão na lista de processos corretos	59

Figura 3.27 - Monitoria representativa de processos e rotativa sobre as máquina do grupo.....	61
Figura 3.28 - O SDF em domínios de tolerância a falhas	62
Figura 4.1 - Relacionamento entre a arquitetura proposta e as classes criadas	66
Figura 4.2 - Diagrama de Classes do núcleo principal do SDF.....	67
Figura 4.3 - Classe ServicoReconfiguracao - métodos utilizados pelos objetos da própria classe...	69
Figura 4.4 - Métodos que são executados pelos objetos do módulo SDF local.....	72
Figura 4.5 - Classe ServicoDFGerente com seus métodos principais.....	75
Figura 4.6 - Classe ServicoDFAgenteConexao e seus métodos	78
Figura 4.7 - Classe ServicoDFAgenteConexao com seu principal método	79
Figura 4.8 - Trecho do código da DLL InvestigaTabelaSO.c	80
Figura 4.9 - Atributos da classe ListaProcessosCorretosAplicacao.....	81
Figura 4.10 - Métodos da classe ListaProcessosCorretosAplicacao.....	82
Figura 4.11 - Classe ListaProcessosFalhosAplicacao com seus principais métodos e os atributos..	84
Figura 4.12 - Classe AnalisadorDeConexao	85
Figura 4.13 - Classe Pinger	86
Figura 4.14 - Classe ListaExcluidosParticionamento	87
Figura 4.15 - Classe ThreadSolicitaInvestigacaoProcessos e classe ServicoDFGerente	87
Figura 4.16 - Classe ThreadMonitoriaServidorNomes	88
Figura 4.17 - Classes ThreadAtualizaHistorico, Classe ThreadHistorico e Classe ServicoReconfiguracao	89
Figura 4.18 - Classe ThreadAtualizaNumeroReconfiguracao, Classe ThreadNumeroEstadoReconfiguracao e Classe ServicoReconfiguracao.....	90
Figura 4.19 - Classe ThreadSolicitaVotos, Classe ThreadVotos e Classe ServicoReconfiguracao .	91
Figura 4.20 - Classe Temporizador e Classe ServicoReconfiguracao	91
Figura 4.21 - ThreadRetiraProcessoGrupo, Classe ThreadSolicitaRetiradaProcessoGrupo, Classe ThreadRetiraProcessosMaquinaFalha e Classe ServicoReconfiguracao	92
Figura 4.22 - Classe ThreadVotosProximoCoord e Classe ServicoReconfiguracao	93
Figura 4.23 - Classe TemporizadorEleicaoCoordenador e Classe ServicoReconfiguracao.....	94
Figura 4.24 - Classe ThreadInsereObjetoParaGerenciamento e Classe ServicoReconfiguracao.	94
Figura 4.25 - Interfaces de monitoria.....	95
Figura 4.26 - SDF monitorando diversos tipos de processos e serviços.	96
Figura 4.27 - Classes que representam os processos monitorados pelo SDF	97
Figura 4.28 - Método <i>respondeMonitoria()</i> da Classe ProcessoInterfaceCorreio	98
Figura 4.29 - Método <i>respondeMonitoria()</i> da Classe ProcessoInterfaceWeb.....	99

Figura 4.30 - Método <i>respondeMonitoria</i> () da Classe <i>ProcessoInterfaceServico</i>	100
Figura 4.31 - Diagrama de classes do Visualizador e as classes <i>ServicoDFGerente</i> e <i>ServicoReconfiguracao</i>	101
Figura 4.32 - Diagrama : Processo responde solicitações fora dos níveis definidos	103
Figura 4.33 - Diagrama : Detecção de processo falho	104
Figura 4.34 - Diagrama : Solicitação de retirada do processo falho	105
Figura 4.35 - Diagrama : Retirada do processo falho do objeto <i>ListaProcessosCorretosAplicacao</i> e inserção no objeto <i>ListaProcessosFalhosAplicação</i>	105
Figura 4.36 - Diagrama : Detecção de falha do módulo SDF de uma máquina com processo suspeito	106
Figura 4.37 - Diagrama : Detecção de indisponibilidade de uma máquina que tem processos monitorados.	107
Figura 4.38 - Diagrama : Ações tomadas quando uma máquina perceber que está sem contato com o Servidor de Nomes.....	109
Figura 4.39 - Diagrama : Atividades executadas quando uma máquina volta ao grupo depois de ter sido particionada.....	110
Figura 4.40 - Diagrama : Primeira fase do protocolo de acordo para retirada do grupo dos processos de uma máquina falha	111
Figura 4.41 - Diagrama : Segunda fase do protocolo de acordo quando há unanimidade sobre a falha de uma máquina.....	112
Figura 4.42 - Organização das referências a objetos do SDF no servidor de nomes	113
Figura 4.43 - IDL do SDF com apenas algumas das operações principais	116
Figura 4.44 - Classe <i>ServicoReconfiguracao</i> herda métodos de classe originadas a partir de uma IDL.....	117
Figura 4.45 - Classes do SDF com seus respectivos Skeleton gerados na compilação da IDL.....	118
Figura 5.1 - Gráfico representando falhas de processos individuais em grupos com quantidades de processos diferentes (3, 7 e 15).....	134
Figura 5.2 - Gráfico das médias de tempo de reconfiguração quando ocorre falha de uma máquina – visão por número de máquinas no grupo.....	135
Figura 5.3 - Gráfico das médias de tempo de reconfiguração quando ocorre falha de uma máquina – visão por número de máquinas no grupo.	135
Figura 6.1 - Domínios de tolerância a falhas com seus grupos de objetos replicados.	140
Figura 6.2 - Componentes definidos na arquitetura de tolerância a falhas FT-CORBA.....	142
Figura 6.3 - Componentes FT-CORBA comparados aos componentes do SDF.....	147

Figura 6.4 - Monitoria de processo que implementa a interface FT-CORBA <i>PullMonitorable</i>	148
Figura 6.5 - Classes de processos interfaces utilizadas no SDF	149
Figura 7.1 - Arquitetura dos serviços GroupPac.....	154
Figura 7.2 - Arquitetura DOORS.....	158
Figura 7.3 - Serviço de monitoração da arquitetura OGS	162
Figura 7.4 - Ferramenta Piranha sobre o Electra	164
Figura 8.1 - Utilização da infraestrutura de detecção de falhas do SDF para replicação de objetos	172
Figura 8.2 - Utilização da infraestrutura de detecção e <i>diagnóstico</i> de falhas do SDF para replicação de objetos	173

Lista de Tabelas

Tabela 1.	Síntese dos dados encontrados após os teste de retirada de processos individuais do grupo (todos os dados).....	124
Tabela 1.1.	Valores encontrados nas amostras da Tabela 1.....	125
Tabela 2.	Síntese dos dados encontrados após os teste de retirada de processos individuais do grupo (excluídos valores extremos).....	126
Tabela 3.	Síntese dos dados encontrados após os testes de retirada de processos de uma máquina que falhou (todos os dados).....	127
Tabela 3.1.	Valores encontrados nas amostras da tabela 3.....	127
Tabela 4.	Síntese dos dados encontrados após os testes de retirada de processos de uma máquina falha (excluídos os valores extremos).....	128

Capítulo 1

Introdução

Neste capítulo apresentaremos a motivação para o desenvolvimento de um Serviço de Diagnóstico de Falhas em CORBA. Mostraremos a estrutura da dissertação e os objetivos pretendidos com este trabalho.

1. Introdução

Sistemas Distribuídos são, geralmente, caracterizados por processos espalhados numa rede de computadores, onde a comunicação entre processos se dá por troca de mensagens. Falhas de processos ou de canais de comunicação, se não forem devidamente tratadas, podem resultar na interrupção das aplicações e, portanto, dos serviços relacionados [Jal94]. Em algumas aplicações (ex.: comércio eletrônico), tais interrupções podem implicar em grandes prejuízos ou perdas. Portanto, aplicações distribuídas precisam dispor de técnicas de Tolerância a Falhas que permitam um serviço continuado na presença de falhas.

Em geral, o tratamento de falhas, seja pelo mascaramento destas ou pela recuperação do estado do sistema, requer o diagnóstico dos componentes falhos, identificando-os e informando aos processos operacionais, de uma forma consistente, o diagnóstico estabelecido. Portanto, o diagnóstico compreende, não somente a detecção de falhas, mas também o estabelecimento e difusão de uma visão coerente sobre o estado de todos os processos monitorados. Essa visão sobre o estado do sistema é conhecida como *síndrome* na literatura relacionada a diagnóstico ao nível do sistema [BB92, RD95, DN98] onde, em geral, se adota o modelo dos sistemas distribuídos síncronos [Fla91] (com tempos conhecidos para propagação de mensagens). Em sistemas assíncronos [FLP85], um diagnóstico mais preciso é geralmente obtido pela combinação de técnicas de detecção que envolvam mais do que mecanismos temporais. A visão coerente do estado dos processos é um requisito fundamental para o funcionamento de técnicas de tolerância a falhas como Replicação Ativa de Componentes [Jal94].

O objetivo de gerenciamento em sistemas distribuídos é assegurar que uma aplicação com processos e serviços remotos funcione adequadamente sobre uma infraestrutura distribuída [VR01]. Com base neste diagnóstico, é possível executar ações preventivas e corretivas a fim de manter a aplicação em funcionamento ou para fazê-la retornar a uma condição aceitável para os usuários.

Esta dissertação apresenta a concepção e desenvolvimento de uma arquitetura distribuída orientada a objetos para diagnosticar falhas de processos de uma aplicação distribuída. Esta

arquitetura especifica um Serviço de Diagnóstico de Falhas (SDF) sobre a plataforma CORBA (*Common Object Request Broker Architecture*) para ambientes assíncronos, conforme apresentado em [BM01]. CORBA é um padrão para desenvolvimento de aplicações em ambientes distribuídos heterogêneos. Outras funcionalidades devem ser adicionadas ao ORB através de um serviço CORBA. A idéia de um Serviço de Diagnóstico de Falhas em CORBA é apropriada pois evita que aplicações distribuídas tratem questões de detecção e diagnóstico dentro do seu código. Isto permite que o desenvolvedor concentre seus esforços no desenvolvimento de sua aplicação, ao passo que se utiliza dos serviços que estão disponíveis no ambiente. Além disso, um serviço de tal natureza possibilita a utilização de técnicas combinadas para detecção de falhas dos processos de uma aplicação.

O objetivo do serviço de diagnóstico, proposto e implementado, é monitorar, continuamente, os processos de uma aplicação distribuída permitindo que os usuários, ou programas clientes, tenham acesso às informações sobre o estado destes processos, bem como o estado das máquinas onde estes processos residem. Permite o acompanhamento de diferentes níveis de tempo de comunicação entre as máquinas e os processos monitorados, a fim de determinar quando estes níveis não estão atendendo aos limites definidos como aceitáveis pela aplicação. Outros serviços que não fazem parte diretamente da aplicação, mas são necessários ao seu funcionamento, como serviços WEB e Correio, também podem ser monitorados.

O serviço de diagnóstico de falhas é distribuído e tolerante a falhas. Foi implementado através de N módulos espalhados numa rede sobre uma plataforma de comunicação CORBA/Java e continuará funcionando mesmo que $N-1$ de seus módulos falhem. Estes módulos comunicam-se trocando informações de diagnóstico.

O SDF inclui um Visualizador que pode ser inicializado em diferentes máquinas e que apresenta informações de diagnóstico. Os Visualizadores apresentam os eventos de falhas, o estado de todos os processos monitorados e o histórico das máquinas do grupo. O Serviço permite também monitorar processos, que foram tidos como falhos por causa de um particionamento na rede (por exemplo um problema em um switch ou em um hub), mas que, depois de resolvido o problema, devem voltar a ser monitorados.

Diferentemente dos sistemas síncronos, nos ambientes assíncronos (Internet, por exemplo), limites de tempo (*timeouts*) não podem ser usados como indicação precisa de falhas e/ou estabelecimento do Diagnósticos [FLP85, Mac00]. Para gerar diagnósticos de falhas mais precisos, o SDF se utiliza de *timeouts* adaptáveis [Mac00], consultas a tabelas do sistema operacional e testes locais sobre o processo suspeito. Para o estabelecimento de uma visão coerente dos processos entre os módulos SDF, foi desenvolvido um protocolo de decisão em duas fases (baseado no *Two-Phase Commit*) com características não bloqueantes. Esta visão coerente é compartilhada pelos módulos SDF que continuarem a manter contato com o servidor de nomes utilizado pelo grupo.

A importância do trabalho está em reunir conceitos pertinentes às áreas de diagnóstico, detecção de falhas e gerenciamento, propondo mecanismos mais precisos para detecção e diagnóstico de processos em sistemas distribuídos assíncronos. Serve como ponto de partida para atender às necessidades de gerenciamento de processos de aplicações distribuídas. Sobre ele é possível desenvolver aplicações de gerência que auxiliem a execução confiável de aplicações com processos e serviços remotos.

O SDF foi implementado e testado num ambiente JAVA/CORBA. Ele faz parte do projeto MONITOR (gerenciamento em sistemas distribuídos) em desenvolvimento no Laboratório de Sistemas Distribuídos – LaSiD/UFBA.

1.1. Estrutura da Dissertação

No capítulo 2, apresentaremos os conceitos de gerenciamento, diagnóstico e detecção de falhas em ambientes distribuídos. Analisaremos o que envolve gerenciamento em Sistemas Distribuídos e qual a relação entre o gerenciamento e diagnóstico.

No capítulo 3, apresentaremos o serviço de diagnóstico de falhas e sua arquitetura distribuída, orientada a objetos. Descreveremos os componentes do serviço de diagnóstico de falhas (SDF), especificado na sua arquitetura, e como estes interagem para atingir seus objetivos.

No capítulo 4, mostraremos como o serviço de diagnóstico foi implementado em Java, utilizando a plataforma CORBA do Visibroker 3.4. Apresentaremos alguns diagramas de classes e de seqüência para mostrar a interação entre os objetos dos módulos SDF e a ordem das ações executadas por eles. Conseqüentemente, este capítulo é particularmente direcionado às pessoas que têm interesse em conhecer detalhes da implementação do serviço.

No capítulo 5, apresentaremos os resultados dos testes executados sobre o serviço e as conclusões que chegamos após os testes.

No capítulo 6, mostraremos o padrão FT-CORBA e como o SDF foi ajustado para estar de acordo com este padrão, no que diz respeito a detecção de falhas.

No capítulo 7, descreveremos alguns trabalhos correlatos que tratam de detecção e diagnóstico sobre plataforma CORBA, comparando as principais características destes com o SDF.

No capítulo 8, concluiremos a dissertação apresentando as contribuições deste trabalho. Mostraremos trabalhos em andamento e perspectivas de trabalhos futuros para dar continuidade ao serviço de diagnóstico.

Capítulo 2

Detecção de falhas, Diagnóstico de Sistema e Gerenciamento em Sistemas Distribuídos

Este capítulo apresenta uma visão geral dos conceitos básicos que deram suporte a criação do serviço de diagnóstico de falhas. Faremos uma breve descrição do que envolve gerenciamento, diagnóstico e detecção de falhas em sistemas distribuídos.

2. Detecção de falhas, Diagnóstico de Sistema e Gerenciamento em Sistemas Distribuídos

Sistema distribuído consiste em uma coleção de computadores autônomos interligados em uma rede de computadores e equipados com software apropriado[CDK96]. É composto por um número finito de máquinas, onde cada máquina possui uma memória local e, eventualmente, uma memória estável. Os processos em cada máquina se comunicam através de troca de mensagens, enviadas pelo canal de comunicação utilizado. Juntos, estes processos formam aplicações distribuídas. O estado global do sistema distribuído consiste em todos os estados locais dos processos mais o estado do canal de comunicação, isto é, as mensagens em trânsito [Gar99].

Em aplicações centralizadas, quando ocorre uma falha no sistema, todo o serviço prestado pela aplicação fica comprometido, enquanto que em aplicações distribuídas é possível contornar a falha, não permitindo que esta impacte na disponibilidade geral dos serviços da aplicação. Para isto é necessário implementar técnicas de tolerância a falhas a fim de que o sistema possa se comportar corretamente mesmo na presença de defeitos¹ de seus componentes distribuídos [Jal94]. O primeiro passo, neste caso, geralmente envolve a detecção da falha ocorrida e o posterior diagnóstico do sistema com a relação dos processos falhos. Com este diagnóstico é possível tomar ações de gerenciamento para preservar a continuidade dos serviços ou para trazer a aplicação ao estado anterior à falha.

Este capítulo dará uma visão geral dos conceitos básicos que deram suporte a criação do serviço de diagnóstico de falhas. Inicialmente apresentaremos, no item 2.1, a terminologia adotada na dissertação e, em seguida apresentaremos os modelos de falhas no item 2.2. No item 2.3 serão descritos, brevemente, alguns modelos de sistemas com suas características principais, e no item 2.4 associaremos os modelos de falhas aos modelos de sistemas descritos. No item 2.5 apresentaremos como ocorre a detecção nos modelos de sistemas. Vamos nos concentrar mais na detecção de falhas do modelo assíncrono, pois este é o modelo para o qual o serviço de diagnóstico foi projetado. No item 2.6 mostraremos o que envolve diagnóstico de sistemas em sistemas assíncronos e no item 2.7 relacionaremos

¹ O termo defeito será explicado no item 2.1 quando apresentaremos a terminologia utilizada na dissertação

gerenciamento com diagnóstico de sistemas. Finalizaremos com um resumo dos conceitos vistos e como estes se relacionam com o serviço de diagnóstico que será apresentado no capítulo seguinte desta dissertação.

2.1. Terminologia Adotada na Dissertação

Como existe ambiguidade na literatura em Português para os termos em inglês *fault*, *error* e *failure*, utilizaremos aqui a seguinte terminologia: falha para *fault*, erro para *error* e defeito para designar *failure*. Estes termos têm significados diferentes do ponto de vista de tolerância a falha. Falha pode ser causada por software (problema de especificação e implementação), por hardware (componentes defeituosos) ou por agentes externos (radiação e interferência eletromagnética, entre outros) [Jal94]. O erro é causado por uma falha. Se existe um erro no sistema, então existe uma sequência de ações que podem ser executadas e que levará a um defeito, a não ser que técnicas de tolerância a falhas sejam empregadas. Um componente é considerado defeituoso quando não atende às especificações para as quais foi projetado.

Como o defeito de um componente sob o ponto de vista do sistema distribuído é uma falha, utilizaremos o termo “detecção de falhas” para significar o mesmo que “detecção de defeitos”. A expressão detecção de falhas é mais utilizada na literatura e mais intuitiva para a maioria das pessoas e por isso utilizaremos esta nomenclatura. No entanto, falamos na arquitetura SDF de um detector de defeitos.

Tradicionalmente, falhas são tratadas descrevendo-se seu comportamento resultante e agrupadas em uma estrutura hierárquica de modelo de falhas.

2.2. Modelos de Falhas

Informalmente, tolerância a falha é a habilidade do sistema se comportar de maneira bem definida, mesmo se falhas ocorrerem. Para isso, o primeiro passo é especificar o tipo de

falha a ser tolerada, ou seja o modelo de falhas que o sistema assume. Modelos de falhas definem comportamento do componente quando há uma falha. Entre estes, podemos citar o modelo de falhas do tipo *crash*, o modelo *crash-recovery*, o modelo *fail-stop*, o modelo bizantino, o modelo de falhas por omissão e o modelo de falhas temporal.

Modelo de falha do tipo *crash* [VR01] ocorre quando o processo pára a sua execução em um ponto específico do tempo e não realiza qualquer computação depois disso. Ele funciona perfeitamente até o momento onde repentinamente morre (*crash*). A diferença deste modelo para o **modelo *crash-recovery*** [VR01] é que no modelo de falhas *crash* o processo que retorna não tem memória de sua existência passada (simplesmente ele foi restartado), enquanto no modelo *crash-recovery* o processo preserva seu antigo estado com, por exemplo, seu último *checkpoint* antes da falha.

No **modelo *fail-stop*** [VR01] um processo pára de funcionar, não emitindo mais qualquer resposta, sendo isso facilmente detectável pelos seus vizinhos. O **modelo bizantino** [JP94] ocorre quando o processo responde a uma solicitação com um comportamento arbitrário, diferente do especificado. Num **modelo de falha por omissão** [JP94], um processo não responde às solicitações temporariamente.

Existe também o **modelo de falhas temporal** [JP94], onde a resposta de um processo não está disponível dentro do intervalo especificado. Uma falha temporal pode acontecer quando a resposta vem muito cedo ou quando esta está atrasada, sendo que neste caso esta falha pode ser denominada de falha de performance [PJ94,VR01,Gar99].

Os modelos de falha estão relacionados aos requisitos estabelecidos nos modelos de sistema, conforme veremos a seguir.

2.3. Modelos de Sistemas

Várias pesquisas em Sistemas Distribuídos têm como objetivo propor modelos de sistemas que atendam às necessidades das aplicações. Naturalmente, em ambientes controlados, é mais fácil ter domínio sobre as variáveis de ambiente, e assim garantir um nível de serviço estável para as aplicações existentes. Em ambientes como a Internet existem muitas

variáveis que não estão sob o controle da aplicação e que podem inviabilizá-la se não forem tratadas adequadamente. Cabe ao modelo formalizar questões referentes aos aspectos temporais nos ambientes. Alguns modelos se adaptam mais a um tipo de ambiente do que a outros [Ver97]. Atualmente, percebe-se que as pesquisas têm se encaminhado para a elaboração de modelos com características híbridas dos dois modelos mais conhecidos e estudados: modelo Síncrono e Assíncrono [Fla91, FLP85]. Estes novos modelos propõem uma solução intermediária no que se refere as características temporais. Entre eles podemos citar: os modelos Quasi-Síncrono [CA98] e o Assíncrono Temporizado [FC98].

A seguir, veremos, resumidamente, as principais características dos modelos síncrono, assíncrono, quasi-síncrono e assíncrono temporizado.

2.3.1. Modelo Síncrono

No modelo de sistemas Síncrono [Fla91] são limitados e controladas todas as variáveis de ambiente. Supõe-se a existência de relógios sincronizados. O atraso na entrega de mensagens, as taxas de desvio dos relógios locais, a velocidade de execução dos processadores e a carga da rede, são conhecidos e limitados dentro do sistema. Desta forma, fica claro que este tipo de modelo se encaixa bem em redes onde é possível ter controle sobre todas as variáveis do ambiente.

Para aplicações que são críticas é necessário garantir que todas as restrições temporais sejam satisfeitas. Para aplicações que desejem ser disponibilizadas em ambientes de comportamento dinâmico, como a Internet, é difícil prever todos os tempos envolvidos para a execução de cada uma de suas atividades. Assim, aplicações projetadas sob o modelo de sistema Síncrono têm dificuldades de oferecer níveis de serviço adequados, quando estão sobre uma infraestrutura não confiável e imprevisível.

Resumindo, as características principais deste modelo são:

- velocidade de processamento dos vários processadores é não nula, e a sua diferença é limitada e conhecida;

- tempo necessário para a disseminação de uma dada mensagem é limitado e conhecido;
- taxa de desvio dos vários relógios locais é limitada e conhecida; e
- noção de tempo global, isto é a diferença entre os relógios locais é limitada e conhecida.

2.3.2. Modelo Puramente Assíncrono

O modelo de Sistemas Assíncrono [FLP85] é naturalmente mais indicado quando se está trabalhando com uma infraestrutura não-confiável, pois este não faz suposição de tempo sob nenhuma das variáveis essenciais. Neste modelo não existem limites definidos para a execução de atividades como, por exemplo, velocidade de processamento, entrega de mensagens, diferenças entre relógios das máquinas envolvidas, taxas de desvio dos relógios locais, etc. Como não se faz suposições de tempo, não é possível distinguir um processo falho de um processo que esteja muito lento.

Aplicações desenvolvidas sob este modelo esperam que um problema consiga ser, em algum momento, resolvido, mesmo que isto demore um período de tempo desconhecido [Mac94]. Como não é possível prever em quanto tempo uma mensagem será entregue a um outro processo cooperante, detectar falhas torna-se uma tarefa difícil. Não ter mecanismos temporais dificulta saber se houve um problema (na rede, na máquina ou no processo) ou se o processo está apenas lento em consequência de uma sobrecarga ou de um tráfego mais intenso na rede.

Em trabalhos anteriores já foi provado que é impossível a obtenção de consenso para o modelo puramente Assíncrono, se este admitir a possibilidade de falhas [FLP85]. Neste caso, quando um elemento não responde, os outros ficam na espera eternamente, não sendo possível atingir o consenso [CT96] e, conseqüentemente, problemas como eleição de líderes e *membership* [Bir93] também não são resolvidos no modelo puramente Assíncrono.

A abrangência deste tipo de modelo é grande, mas a aplicabilidade fica comprometida quando se admite a existência de falhas.

Resumo das características do modelo de sistema Assíncrono:

- serviços não tem especificação de tempo, ou seja, suas especificações descrevem quais as saídas e os estados transitórios que devem ocorrer em resposta a entradas, mas não especificam limites de tempo para a sua ocorrência;
- como, neste modelo, um observador não tem como distinguir um processo não falho, mas muito lento, de um processo falho, a maior parte dos serviços que tem importância na prática como consenso, eleições de líderes ou membership não são implementáveis de forma direta, embora existam soluções probabilísticas e com uso de detectores de defeitos [BHM99, HMRT99]; e
- é de especial interesse, pois soluções propostas para ele são aplicáveis aos outros modelos, já que sua especificação não incorpora restrições temporais.

2.3.3. Modelo Quasi-Síncrono

O modelo Quasi-Síncrono [Alm98, AV96, VA95] situa-se entre os modelos Síncrono e Assíncrono, sendo que suas características principais aproximam-se das características encontradas no modelo Síncrono. Propõe um *framework* que contém especificações temporais para *alguns* aspectos do sistema. Neste Modelo apenas uma parte do sistema tem características síncronas, sendo que a outra parte pode exibir um comportamento mais incerto.

Para os requisitos do sistema são estipulados limites artificiais que tem pouca probabilidade de serem violados. Há, todavia, alguns limites que podem ser alterados ou “relaxados” durante o funcionamento do sistema, como a propriedade de entrega de mensagens. Neste caso, os demais limites definidos (atrasos de processamento, imprecisão de relógios locais e diferenças entre os relógios das máquinas) acontecem num intervalo de tempo especificado sob pena de gerar uma falha temporal.

O modelo Quasi-Síncrono foi formulado tendo em mente aplicações de tempo real que possam permitir o não cumprimento dos requisitos temporais, desde que as situações onde estes prazos não são cumpridos sejam controladas. Para isto utiliza-se a noção de envelopes de qualidades de serviço (QoS) e detecção de falhas temporais. Neste modelo é possível

definir níveis de qualidade de serviço a serem oferecidos à aplicação. Quando, devido a ocorrência de falhas ou numa situação de sobrecarga, não é possível continuar com uma determinada qualidade de serviço, pode ser vantajoso mudar para um outro nível de qualidade de serviço. Deste modo a aplicação continuará a fazer progresso, ainda que de uma forma degradada (*graceful degradation*).

O modelo Quasi-Síncrono utiliza dois tipos bem definidos de canais de comunicação: um canal de controle Síncrono (a ser utilizado para detecção de falhas temporais) e um canal genérico para a troca de mensagens. A idéia fundamental é utilizar o canal síncrono para transportar informação vital que será utilizada para validar o restante do protocolo.

Resumindo, as características principais deste modelo são:

- são limitados: o atraso de processamento, a imprecisão dos relógios locais e a diferença entre os relógios das máquinas;
- é possível definir níveis de qualidade de serviço a serem oferecidas a aplicação; e
- tem dois tipos bem definidos de canais de comunicação: um canal de controle Síncrono (utilizado pelos participantes para controle) e um canal genérico para a troca de mensagens.

2.3.4. Modelo Assíncrono Temporizado

O modelo Assíncrono Temporizado [FC98] situa-se entre os modelos Assíncrono e Síncrono, sendo que suas características principais aproximam-se das características encontradas no modelo Assíncrono. O núcleo do modelo assume a existência de um serviço datagrama, um serviço de gerenciamento de processo e acesso a relógios locais (de hardware). Consistem de um conjunto de processos P conectados por um serviço datagrama e um serviço “*broadcast*” (implementado sobre o serviço datagrama). Estes serviços podem falhar em entregar a mensagem no tempo estabelecido e podem perder mensagens. Processos executam em nodos da rede. Dois processos são ditos remotos se eles executam em nodos distintos, de outra forma são ditos locais. Cada processo acessa seu relógio local.

O serviço de gerenciamento de processos, que roda em cada nodo, utiliza este relógio para gerenciar alarmes quando um processo local requisita ser acordado. Considera-se que os sistemas possuem intervalos em que estão estáveis, e funcionam como sistemas síncronos, e intervalos pequenos nos quais possuem instabilidade, e, portanto, características assíncronas. Sempre, depois de um período instável, há um intervalo de tempo onde o sistema se mantém estável. Se, dentro de um intervalo I , a maioria dos processos se mantiverem corretos, comunicando-se dentro dos tempos esperados, pode-se dizer que o sistema está “*majority-stable*” no intervalo I . Os relógios locais são acertados entre si através de algoritmos de sincronização, e acertados com o tempo real, utilizando GPS, por exemplo. Assim, os relógios são ditos estáveis, nunca se desviando mais do que a taxa máxima prevista. Desta forma é possível construir serviços temporizados que avisam quando as características desejadas não podem ser cumpridas. Um serviço é dito temporizado (*timed*) quando um intervalo de tempo é dado dentro do qual este serviço deve responder às solicitações feitas a ele. Em função do desvio dos relógios locais e dos tempos de processamento e comunicação, são definidos limites máximos de atraso permitidos.

Resumo das características do modelo de sistema Assíncrono Temporizado:

- define o serviço de gerenciamento de processos, serviço de relógio de hardware e o serviço datagrama;
- todos os serviços são temporizados;
- processos acessam relógios de hardware que são próximos ao tempo-real e que nunca se desviam mais do que um valor conhecido e limitado ρ ; e
- não existe um limite no atraso de transmissão.

2.4. Associando Modelos de Sistema e Modelos de Falha

Nos sistemas síncronos as falhas podem ser divididas em quatro categorias, de acordo com os modelos de falha apresentados anteriormente: *fail-stop*, omissão, temporal e bizantina.

Nos sistemas assíncronos, as falhas podem ser *crash*, omissão, bizantina e *crash-recovery* [Cir99].

No modelo quasi-síncrono, existem todos os tipos de falhas do modelo síncrono, acrescido das falhas temporais, que acontecem quando há violação dos níveis de qualidade de serviço propostos pela aplicação [AV96].

Nos sistemas assíncronos temporizados, quanto à comunicação entre os processos, as falhas admitidas são por omissão ou performance (tempo de processamento ou atraso na transferência da mensagem maior do que o limite máximo definido). Por sua vez, as falhas de processos podem ser *crash* ou de performance [FC98].

2.5. Detecção de Falhas nos Modelos de Sistemas

Sistemas distribuídos tolerantes a falhas são projetados para prover serviços contínuos e confiáveis, ainda que ocorram falhas em seus processos. Um componente básico destes sistemas é o detector de defeitos [TC00]. Um detector de defeitos provê informação sobre quais processos falharam. Esta informação é, tipicamente, dada sob a forma de uma lista de processos ou componentes suspeitos.

A detecção de falhas é definida de acordo com o modelo de sistemas ao qual deve ser aplicada. Para os *sistemas síncronos*, a detecção de falhas acontece através do uso de um detector de defeitos perfeito, que utiliza mecanismos de *timeout* em suas consultas [CGS00]. Este tipo de detector suspeita somente de processos falhos e, em algum momento, todos os processos falhos serão suspeitos.

Nos *sistemas quasi-síncronos* é definida uma nova classe de detectores de defeitos, o serviço de detecção de falhas temporais (SDFT). Existe uma cópia deste serviço de detecção em cada nodo. Funciona como um oráculo que é utilizado pelos protocolos de comunicação. Este serviço, ao ser executado utilizando canais especiais (síncronos), fornece os meios necessários para construir protocolos que conseguem obter as suas propriedades de segurança num tempo limitado e conhecido num sistema não totalmente síncrono [AV96] [Alm98].

Nos *sistemas assíncronos temporizados* existem os detectores *fail-aware* [FC98]. É necessário estender o modelo com algumas suposições “*progress assumption*”, para poder implementá-los. Estes detectores são distribuídos, implementados por um conjunto de módulos locais, um para cada processo. Trabalham de forma integrada com o processo ao qual ele está associado. Cada módulo mantém o conjunto de processos que ele suspeita de terem falhado. A propriedade *fail-awareness*² é implementada utilizando periódicas mensagens de “*I am alive*” dos processos. Cada processo mantém uma lista de processos estáveis, “*Alive*” que contém todos os processos dos quais ele recebeu recentemente uma mensagem de “*I am alive*” (ditos estáveis). Na sua mensagem de “*I am alive*” o processo inclui esta lista. O protocolo assegura que nenhum processo q pode suspeitar de um processo estável p mesmo que p e q não possam se comunicar durante um período de tempo. Para assegurar isto, o processo q somente poderá suspeitar do processo p se a maioria dos processos não incluírem p em suas mensagens de “*I am alive*”. A propriedade requer que um módulo detector de defeito suspeite de seu próprio processo se a maioria dos processos suspeitarem. Assim, se um processo p é suspeitado pela maioria dos processos, então o detector *fail-aware* de p pode avisá-lo que ele é suspeito de falha. Como os processos incluem, em sua mensagens de “*I am alive*”, todos os processos considerados como sendo estáveis, em algum momento nenhum processo pode suspeitar de um processo estável.

Em *sistemas puramente assíncronos*, é impossível detectar precisamente a falha de um processo, já que não é possível distinguir entre processos falhos e lentos. Chandra e Toueg estenderam o modelo puramente assíncrono com a noção de detectores de defeitos para resolver problemas clássicos, como consenso e *membership*[CT96]. Estes mecanismos de detecção de falhas podem cometer erros, e, por isso, são chamados de detectores não-confiáveis de defeitos para sistemas com falha do tipo *crash*. Eles simplificam a tarefa de projetar algoritmos para este tipo de sistema por encapsularem a noção de tempo. Modelos utilizando esses detectores não-confiáveis não podem ser ditos puramente assíncronos, já que eles produzem a ilusão de um sistema assíncrono através do encapsulamento de todas as referências de tempo [Gar99]. Formalmente, estes detectores de defeitos são definidos por duas propriedades: exatidão (*accuracy*) e completude (*completeness*). A propriedade

² A expressão “o processo p suspeita de q ” equivale dizer que “o módulo do detector *fail-aware* de p suspeita de q ”

accuracy é uma propriedade *safety* e garante que, se uma falha é reportada, então ela efetivamente ocorreu. Em sua forma mais fraca, um detector de defeitos nunca suspeitará de pelo menos um processo correto. Esta propriedade é chamada de *weak accuracy*. Como esta propriedade é difícil de ser alcançada, é exigido que ela, em algum momento, seja atingida. Assim, em um detector de defeitos *eventually weak* existe um tempo após o qual algum processo correto não é mais suspeito. De fato, este detector pode cometer muitos erros em prever os estados funcionais dos processos, mas é garantido que ele, com o tempo, não cometerá erros quando se refere a pelo menos um processo correto. No entanto, apenas a propriedade de *weak accuracy* não garante a suspeita de um processo – ela apenas proíbe que o mecanismo de detecção suspeite incorretamente de um processo correto. Por isto, faz-se necessária a definição da segunda propriedade, *completeness*. Esta propriedade garante a característica de *liveness* e estabelece que toda falha no sistema distribuído será finalmente reportada para os demais processos. Informalmente, esta propriedade requer que todo processo falho seja, em algum momento, suspeito por algum processo correto. A propriedade de *strong completeness* estabelece que: todo processo falho será permanentemente suspeito por todo processo correto.

Os detectores de defeitos, propostos por Chandra e Toueg, são agrupados em oito diferentes classes, de acordo com suas propriedades de *completeness* e *accuracy* [CT96]. A classe de detectores $\langle \rangle S$, caracterizada pelas propriedades *eventually weak accuracy* e *strong completeness*, é a classe mais fraca sobre a qual problemas, como consenso, podem ser resolvidos. A propriedade de *strong completeness* pode ser alcançada através do uso de *timeouts* e mensagens do tipo *heartbeat*. Entretanto, uma propriedade *accuracy* (qualquer que seja) pode ser, em um sistema puramente assíncrono, apenas aproximativa [LAF99]. Consequentemente, se o modelo utilizado é assíncrono, a suspeita de falhas é, no melhor dos casos, aproximativa.

Em [Mac00] é proposto um mecanismo, Indicador de Tempo de Conectividade (CTI), utilizado para implementar as propriedades requeridas para os detectores de defeitos da classe $\langle \rangle S$. O tempo de conectividade, *ct*, entre dois processos P_i e P_j , é definido como o tempo que uma mensagem leva para trafegar de P_i a P_j (ou vice-versa) num dado momento.

Em sistemas como a Internet, o tempo de conectividade pode assumir valores distintos dentro de um intervalo de tempo. Ele pode variar de 0 (se $i = j$) até infinito (se P_i e P_j estão atualmente desconectados). O CTI é definido como um mecanismo capaz de fornecer dinamicamente o tempo de conectividade, também chamado de tempo de comunicação.

Como é impossível prever precisamente o futuro, o CTI sugere o tempo de comunicação atual. Assim, existirá um módulo CTI sendo executado em cada máquina do sistema distribuído e ele estará atualizando constantemente as informações sobre conectividade dos processos locais e remotos. Ele também faz investigações no sistema operacional para saber o estado de processos suspeitos. Detectores de defeitos usando CTI assumem um sistema distribuído onde os processos se comunicam através de canais confiáveis (isto é, uma mensagem chegará, em algum momento, ao destino, se o processo estiver em estado correto). Processos falham por falhas do tipo *crash* e *crash-recovery* (neste caso, o processo tem de se manter correto entre as suas falhas durante um tempo suficiente para que a propriedade *eventually weak accuracy* seja atingida). Não se assume qualquer limite na comunicação. A idéia é que o mecanismo CTI seja incorporado ao detector de defeitos para indicar com mais precisão o estado de um processo em sistemas assíncronos.

Em abril de 2000, a OMG lançou a especificação FT-CORBA para introduzir aspectos de tolerância a falhas no padrão CORBA [OMG00]. Entre as áreas contempladas nesta especificação está o gerenciamento de falhas, com a definição das interfaces de detecção de falhas, notificação e análise. Como a arquitetura do SDF já estava definida por nós, este padrão não influenciou diretamente no desenvolvimento do serviço de diagnóstico. No entanto, no capítulo 6 apresentaremos este padrão e as adaptações que fizemos no serviço de diagnóstico para torna-lo de acordo com o FT-CORBA no que se refere as atividades de detecção de falhas.

2.6. Diagnóstico de Sistemas

Sistemas que necessitam de confiabilidade devem utilizar-se de mecanismos para continuar a prover seu serviço mesmo na presença de falhas. Este tipo de sistema não pode utilizar-se apenas de técnicas de prevenção de falhas [Lap85]. Eles necessitam de técnicas específicas

para tolerar as falhas que porventura ocorram. Se defeitos ocorrerem em alguns componentes, um sistema tolerante a falhas deve evitar que haja um defeito geral do sistema, ou seja, que esta situação de falha seja refletida no seu comportamento externo.

Entre as técnicas utilizadas para dar suporte ao desenvolvimento de sistemas tolerantes a falhas podemos destacar Diagnóstico de Sistema [BB92]. O objetivo desta técnica é determinar o conjunto de nodos corretos e falhos, ou seja, determinar o diagnóstico do sistema. Este diagnóstico compreende duas fases. A primeira fase envolve a detecção dos nodos falhos, através de testes. A segunda fase envolve a difusão dos resultados dos testes, através de trocas de resultados entre os nodos corretos. Com a difusão dos resultados dos testes, a síndrome da rede é estabelecida, e todos os nodos corretos têm conhecimento do diagnóstico atual do sistema. Assume-se que os nodos corretos são confiáveis, isto é, eles executam os testes e difundem os resultados com precisão. Os testes são executados em intervalos de tempo bem definidos. Os nodos falhos simplesmente deixam de funcionar, não respondendo aos testes. Assim, diagnóstico é a determinação da situação de falha na rede dada pela sua síndrome.

A pesquisa na área de diagnóstico de sistema recebeu considerável atenção no passado [PRA96, JAL94]. Estes trabalhos fixavam-se em diagnóstico de processadores (e outros sistemas de hardware), frequentemente utilizando um nodo central para receber os testes e executar o diagnóstico. Sistemas Distribuídos necessitam de algoritmos de diagnóstico distribuídos, ou seja que permitam a cada um de seus nodos corretos determinarem quais as unidades falhas e quais as unidades corretas existentes no sistema, sem a necessidade de um nodo central para executar esta tarefa. Recentemente, vários trabalhos tem sido feitos para desenvolvimento de algoritmos de diagnóstico para sistemas distribuídos [RD95, DN98]. Em [Bat00] alguns algoritmos de diagnóstico foram analisados. Geralmente, estes algoritmos concentram-se em como atingir de forma eficaz os nodos para realização de testes de detecção, dentro de um tempo específico, e em como estabelecer uma visão coerente sobre os nodos falhos entre os nodos corretos do sistema. Em [DBA99] foram propostos mecanismos para melhorar a latência dos algoritmos de diagnóstico distribuídos. No entanto, as técnicas utilizadas nos testes para detecção de falhas ficam, em geral, relegadas a simples *timeouts* nestes algoritmos. Infelizmente, este tipo de técnica para desenvolver sistemas tolerantes a falhas requer um ambiente com características síncronas,

não podendo ser aplicadas diretamente em sistemas distribuídos onde o tempo de transmissão de mensagens e o tempo de processamento não são conhecidos [Mac00]. A proposta do desenvolvimento de detectores de defeitos, utilizando mecanismos como CTI, são adequados para utilização na primeira fase do diagnóstico de sistemas em sistemas assíncronos [Mac00]. Diagnósticos que incluam acordo para estabelecimento de uma visão coerente compartilhada entre os nodos são essenciais para este tipo de ambiente uma vez que não há como se garantir que processos distintos cheguem ao mesmo diagnóstico.

Apesar de na literatura diagnóstico de sistema normalmente estar ligado a nodos/processadores, esta técnica pode ser aplicada de forma igualmente válida para um sistema distribuído de processos [Mac00]. Neste caso, o diagnóstico a ser executado refere-se aos processos de uma aplicação distribuída e, conseqüentemente, aos nodos onde eles estão. Assim, detecções mais precisas, juntamente com técnicas para estabelecimento de visão compartilhada apropriadas para ambientes assíncronos, implicam em um diagnóstico de processos adaptado para ambientes sem limites temporais.

2.7. Gerenciamento

De forma genérica, gerenciamento de sistema refere-se ao conjunto das ações de planejar, supervisionar e controlar para prover um serviço adequado e contínuo. Normalmente, na literatura, o termo gerenciamento está associado a recursos em um rede de computadores. O gerenciamento de redes está relacionado com a infraestrutura (computadores / redes de telecomunicação) e como a informação trafega. Dentro deste contexto, é possível observar a existência de gerentes (*managers*), que executam as ações de gerenciamento, e os objetos gerenciados (*managed objects*), que respondem as operações requisitadas. Para que um objeto possa ser gerenciado é necessária a existência de uma interface de gerenciamento, sobre a qual diferentes ações podem ser solicitadas.

Foram definidas pela ISO (*International Organization for Standardization*) cinco áreas funcionais que representam o gerenciamento de redes [X.701]. Cada uma destas áreas tem funções específicas que incluem, entre outras, o controle sobre os objetos gerenciados

(Gerência de Configuração), a manutenção e o acompanhamento do estado de cada um dos objetos (Gerência de Falhas), a monitoria de quais recursos estão sendo utilizados (Gerência de Contabilização), a coleta e interpretação das medições periódicas dos indicadores de desempenho (Gerência de Desempenho) e a proteção e controle de acesso as informações de gerenciamento (Gerência de Segurança). No entanto, nas redes, além da própria infraestrutura e dos recursos de hardware, existem aplicações que também precisam ser assistidas para garantir um funcionamento correto dos serviços aos usuários.

Gerenciamento em sistemas distribuídos, entre outras coisas, tem como objetivo assegurar que aplicações distribuídas executem corretamente sobre uma infraestrutura distribuída [VR01]. Significa monitorar processos e serviços que compõem uma aplicação distribuída, detectando falhas e tomando decisões baseadas nas informações coletadas. Neste caso, os processos e serviços são os objetos gerenciados (*managed objects*) e a aplicação de gerenciamento representa o gerente (*manager*). Assim, pode-se dizer que gerenciamento envolve inicialmente o diagnóstico do sistema [BM01]. Com o conhecimento do diagnóstico do sistema, é possível tomar ações de gerenciamento sobre os processos de uma aplicação distribuída. Estas ações incluem divulgar o estado dos processos da aplicação, reinicializar processos que falharam, finalizar processos, migrar processos de uma máquina para outra, etc.

O gerenciamento das condições da rede associadas às aplicações distribuídas que rodam sobre ela tem sido focado em diversos trabalhos [EM98, NM98, CS97]. Em [BAN97] foi proposta a utilização do padrão CORBA para implementar aplicações de gerência na área de redes. CORBA, pelas suas características, é o padrão adequado para construção de aplicações de gerenciamento para sistemas distribuídos.

2.8. Síntese dos Conceitos Apresentados

Não importa quão bem projetado foi um sistema, sempre haverá a possibilidade de defeitos ocorrerem se falhas forem frequentes ou sérias. A primeira ação para tolerar falhas de uma aplicação é definir o tipo de falha que será tolerada. O modelo de sistemas que a aplicação

estará trabalhando, junto com a semântica de falhas adotada, delimita o estudo de soluções para os problemas provenientes de falhas de processos de uma aplicação. As técnicas utilizadas para detecção de falhas são definidas de acordo com o modelo de sistemas sobre o qual a aplicação vai funcionar. O diagnóstico do sistema depende dos resultados obtidos na detecção e na difusão destes resultados entre as máquinas. Em sistemas assíncronos, o diagnóstico também deve envolver um acordo entre as máquinas para retirada de processos falhos. Isto se dá pois não é possível determinar precisamente falhas de processos remotos, como acontece em sistemas síncronos com o uso de *timeouts*. Com base no diagnóstico do sistema, ações de gerenciamento podem ser tomadas sobre os processos da aplicação. Gerenciamento em sistemas distribuídos envolve o conhecimento dos estados dos processos da aplicação. O objetivo é perceber quando ocorrem falhas (e eventuais mudanças no comportamento da aplicação) para adequar a aplicação a situações de instabilidade. As ações de gerenciamento compreendem desde a visualização de todo o conjunto de processos até ações preventivas ou corretivas, podendo incluir ações de reinicialização dos processos que falharam e finalização dos processos corretos.

No próximo capítulo descreveremos o serviço de diagnóstico de falhas, sua arquitetura e componentes.

Capítulo 3

Serviço de Diagnóstico de Falhas sobre CORBA e sua Arquitetura

Apresentamos, neste capítulo, o Serviço de Diagnóstico de Falhas, sobre plataforma CORBA, e sua arquitetura distribuída, orientada a objetos.

3. Serviço de Diagnóstico de Falhas

O Serviço de Diagnóstico de Falhas (SDF) é um serviço para detectar e diagnosticar processos que compõem uma aplicação distribuída [BM01]. É composto por módulos distribuídos em máquinas da rede que têm processos a serem monitorados. Ele implementa o mecanismo de CTI, proposto em [Mac00], para executar detecções mais precisas e trabalha com falhas do tipo *crash*. Executa as duas fases de um diagnóstico: detecta os componentes falhos e reconfigura o grupo para garantir uma visão única entre seus módulos quando há eventos de falhas. Funciona sobre os conceitos de gerenciamento, diagnóstico e detecção de falhas em aplicações distribuídas, conforme será visto no item 3.1.

Na comunicação entre processos distribuídos de uma aplicação é possível ocorrer falhas na infraestrutura de comunicação (incluindo congestionamento), nos próprios processos ou ainda nas máquinas onde estes residem. O serviço de diagnóstico implementa técnicas de detecção para identificar a falha ocorrida e, em seguida, tomar as ações para retirar o componente falho do grupo de componentes que podem ser utilizados pela aplicação.

Situações de falhas são identificadas através do acompanhamento dos Tempos de Comunicação (TC) entre o SDF e os processos monitorados. Níveis de TC são propostos pelos usuários para a comunicação entre o processo em questão e os módulos SDF. Estes níveis são monitorados pelo SDF periodicamente para informar aos usuários do serviço as falhas que ocorreram e a qualidade de serviço disponível no momento. O SDF também detecta falhas nos seus próprios módulos distribuídos, sendo portanto um serviço tolerante a falhas de seus próprios componentes.

Este capítulo apresenta o serviço de diagnóstico de falhas proposto e desenvolvido nesta dissertação e sua arquitetura. No item 3.1 mostramos os conceitos que deram suporte ao desenvolvimento de um serviço de tal natureza. No item 3.2 apresentaremos a arquitetura do SDF e seus componentes, mostrando como estes interagem. No item 3.3 justificaremos a adoção do padrão CORBA como plataforma de comunicação e no item 3.4 mostraremos como é possível utilizar o serviço para monitorar falhas em aplicações distribuídas. Os tipos

de diagnósticos executados pelo SDF são detalhadas no item 3.5. No item 3.6 descreveremos o protocolo de acordo para retirada de máquinas ou módulos falhos do grupo. O Visualizador, que reporta as informações de detecção e diagnóstico, é apresentado no item 3.7 e no item 3.8 mostraremos como a forma de monitoria do SDF pode ser estendida para atender questões de escalabilidade quando se trata de aplicações complexas. Finalizaremos o capítulo com um resumo do serviço apresentado, item 3.9.

3.1. Estrutura Conceitual do Serviço de Diagnóstico de Falhas

A relação entre gerenciamento, diagnóstico de sistemas e detecção de falhas, segundo nossa visão, foi apresentada em [BM01]. A figura 3.1 retrata esta relação, mostrando que o SDF foi idealizado sobre estes conceitos. Através desta figura é possível perceber que gerenciamento em sistemas distribuídos (de aplicações distribuídas) envolve primeiramente um diagnóstico dos processos. O diagnóstico, por sua vez, é obtido através da detecção de falhas. Para detectar a falha de um processo é necessário monitorá-lo e, por isso, é preciso definir uma interface para monitoria e gerenciamento dos processos. Por causa das características assíncronas do ambiente que está sendo considerado, o diagnóstico necessita de um acordo para o estabelecimento de uma visão coerente (diagnóstico único) entre os processos. Esta visão coerente, que pode não corresponder a realidade, é igual para todos os processos que participam do acordo. Este acordo também está representado na figura 3.1. A segunda etapa do gerenciamento acontece pela atuação direta sobre os processos e serviços gerenciados, como mostra também a figura.

O SDF foi desenvolvido para monitoria de aplicações distribuídas considerando a visão de gerenciamento, diagnóstico e detecção, em sistemas com características assíncronas, apresentada na figura 3.1.

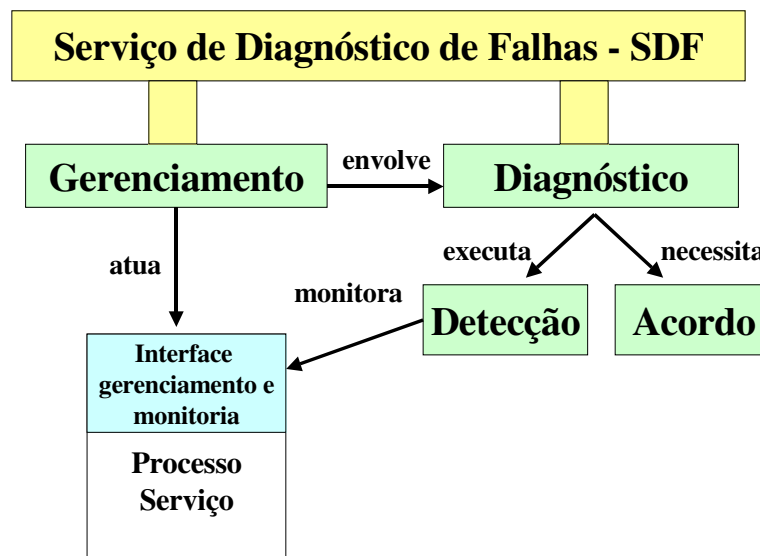


Figura 3.1– Gerenciamento e Diagnóstico de Sistemas

3.2. Arquitetura do Serviço de Diagnóstico de Falhas e seus Componentes

Os módulos do SDF, inicializados nas máquinas, trabalham de forma integrada para executarem detecções de falhas e para manter uma visão consistente do estado dos processos monitorados. O primeiro módulo SDF do grupo, ao ser instanciado, informa o identificador da aplicação à qual ele estará vinculado. A partir deste momento, processos da aplicação podem ser incluídos para monitoria (formando um grupo) se existir um módulo SDF na máquina onde este reside.

A figura 3.2 apresenta módulos SDF em máquinas da rede onde existem processos monitorados (P1, P2, P3, P4, P5 e P6) que fazem parte de uma aplicação distribuída **P**.

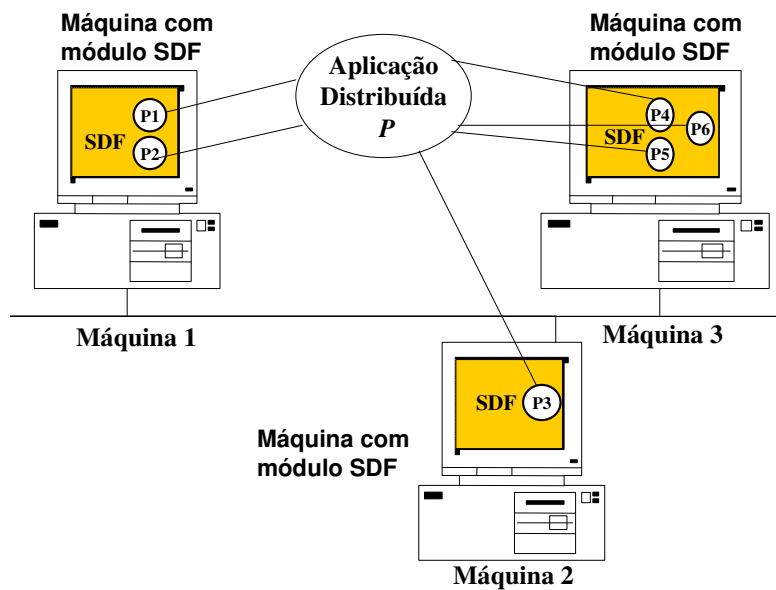


Figura 3.2 - Módulos SDF em máquinas da rede

Cada módulo SDF, em uma máquina, é dividido em dois sub-módulos que trabalham de forma coordenada: o sub-módulo de *Detecção de Falhas* e o sub-módulo de *Reconfiguração*, conforme pode ser visto na figura 3.3.

O sub-módulo de *Detecção de Falhas*, como o próprio nome diz, atua como monitor do tempo de comunicação e detector de defeitos dos processos que compõem uma aplicação. A informação de uma falha e a identificação do seu tipo é gerada por este sub-módulo.

O sub-módulo de *Reconfiguração* é responsável por inserir processos para serem monitorados, bem como retirar os processos detectados como falhos.

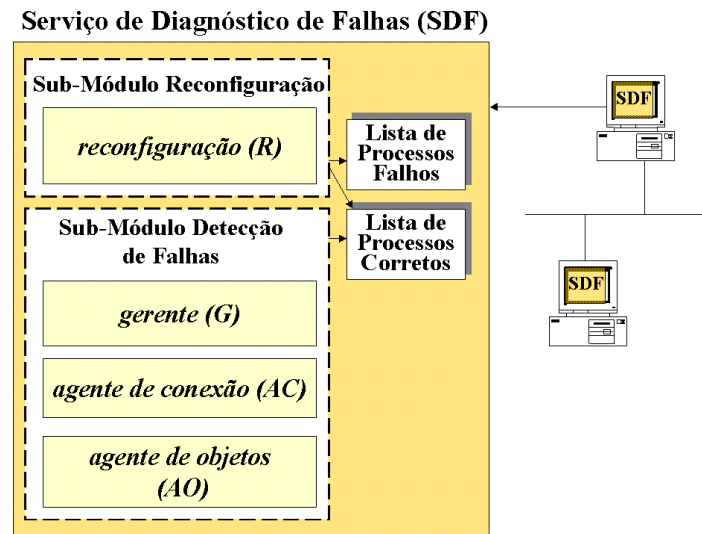


Figura 3.3 – SDF e seus sub-módulos e componentes

Analisando mais detalhadamente a figura 3.3, verifica-se que o componente principal do sub-módulo de reconfiguração é o objeto *reconfiguração (R)*, que mantém a lista de processos disponíveis para uso da aplicação distribuída (considerados corretos). Quanto ao sub-módulo de detecção de falhas é possível ver que este é composto pelo objeto *gerente (G)*, o objeto *agente de conexão (AC)*, e o objeto *agente de objetos (AO)*.

O *agente de conexão* investiga as máquinas que têm processos monitorados, analisando periodicamente o tempo de comunicação entre o módulo do SDF e os processos (locais ou remotos). Os agentes de conexão monitoram todos os processos da aplicação distribuída com os quais estão relacionados. As suspeitas de falhas, assim como a identificação de seus tipos, são geradas pelo agente de conexão durante essa monitoria.

O *gerente* é acionado pelo agente de conexão quando é detectado que o processo monitorado não responde dentro dos níveis de tempo de comunicação propostos. O gerente é informado também sobre o tipo de falha que foi diagnosticada, se é falha da máquina ou do próprio processo. A depender do tipo de falha, ações diferentes devem ser tomadas, conforme será explicado no item 3.5.

O *agente de objetos* tem como função investigar o estado de um processo suspeito que reside na sua máquina. É ele que determinará se houve uma falha individual do processo.

Para realizar as atividades de detecção de falhas e de diagnóstico, os dois sub-módulos utilizam uma estrutura que contém a *lista de processos corretos* da aplicação. Além desta estrutura, o sub-módulo de reconfiguração mantém também a *lista de processos falhos* da aplicação. Estas estruturas podem ser visualizadas na figura 3.3.

3.3. Adoção da plataforma CORBA para comunicação

Como uma forma de tratar e simplificar a computação em sistemas distribuídos heterogêneos, o padrão CORBA (*Common Object Request Broker Architecture*) foi proposto [OMG96]. O CORBA é uma tecnologia padronizada, orientada a objetos (OO), independente de plataforma, e independente de linguagem de programação.

O SDF foi desenvolvido baseando-se na concepção de serviços CORBA (*Common Object Request Broker Architecture*) [OMG96, YD96, Sch97, Kea97]. Um serviço CORBA é um conjunto de objetos e suas interfaces IDL (*Interface Definition Language*) que podem ser invocados por um cliente através do ORB (*Object Request Broker*).

A principal vantagem de um serviço CORBA é que ele não está relacionado a nenhuma aplicação específica. Ele é um bloco básico de construção (*building blocks*), provido pelo ambiente CORBA para qualquer aplicação. A idéia de um Serviço CORBA de Diagnóstico de Falhas é interessante por evitar que as aplicações distribuídas se preocupem com questões de detecção e diagnóstico dentro do seu código. Além disso, um serviço CORBA se beneficia das vantagens oferecidas pela própria plataforma, como interoperabilidade, código orientado a objetos, independência da linguagem de programação, etc.

Os objetos do SDF são objetos CORBA que se comunicam através de um servidor de nomes³ CORBA. Neste servidor estão as referências aos objetos dos módulos SDF. Para se comunicar com um objeto é necessário solicitar ao servidor de nomes a sua referência e, em seguida, executar sob esta as solicitações desejadas.

³ O Serviço de Nomes de CORBA define uma estrutura para associar nomes a objetos remotos definidos na arquitetura. A estrutura definida é uma hierarquia (ou árvore), onde cada ramo define um contexto distinto e cujas folhas são os nomes dos serviços disponibilizados. Assim, a referência completa para o nome de um serviço é dada pelo contexto (os nomes dos nós intermediários) e pelo nome do serviço.

No momento da inicialização do módulo SDF em uma máquina, seus objetos principais são cadastrados no servidor de nomes. Quando objetos de um segundo módulo são inseridos no servidor de nomes, é solicitado ao módulo do SDF, já cadastrado, que envie o estado de todos os processos monitorados pelo serviço (isto é, o estado do grupo).

A figura 3.4 apresenta o módulo SDF da máquina 2 sendo inserido no grupo. Inicialmente ele solicita inscrição de seus objetos no servidor de nomes. Em seguida, recebe as referências aos objetos do módulo SDF da máquina 1, que já estavam cadastrados. Com estas referências é possível solicitar o estado dos processos do grupo. No capítulo 4, item 4.6, veremos como foi implementada esta interação dos objetos através do uso de um servidor de nomes CORBA.

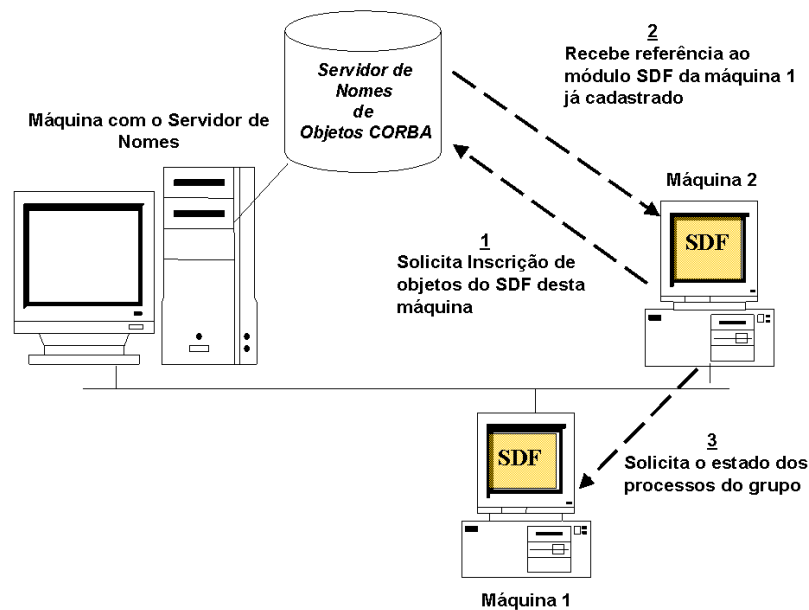


Figura 3.4 - Inscrição do módulo SDF no Servidor de Nomes CORBA

Como pode-se observar, o servidor de nomes é um ponto único de falhas. Existem trabalhos que propõem soluções para um Servidor de Nomes Tolerante a Falhas, como pode ser visto em [LFFOR99]. Essas soluções podem ser usadas para aumentar a confiabilidade do servidor de nomes usado pelos módulos do SDF.

3.4. Monitoração de Aplicações Distribuídas Pelo SDF

Para que um módulo SDF monitore processos remotos de uma aplicação distribuída é necessário que estes processos implementem uma interface de monitoria. Esta interface, deve possuir métodos que permitam a um processo ser inserido na lista de processos corretos e que possibilitem ao SDF verificar o seu estado.

Algumas aplicações necessitam de outros serviços para continuar operando, além de seus processos remotos. Por exemplo, uma aplicação distribuída pode necessitar de uma consulta a um servidor WEB para seu processamento. Esta aplicação pode necessitar também de um serviço implementado em JAVA ou mesmo de um serviço de Correio. Neste contexto, estes serviços também são componentes que necessitam ser monitorados, visto que sua indisponibilidade pode afetar todo o funcionamento da aplicação. No entanto, não é possível fazer isso diretamente, pois eles são produtos que não implementam a interface de monitoria do SDF. Para atender a esta necessidade, propomos a criação de um objeto intermediário, chamado de processo interface, que faz a comunicação entre os serviços que não implementam a interface de monitoria e o SDF.

O objetivo de usar processos interfaces é permitir que estes serviços também sejam acrescentados ao grupo de processos monitorados pelo serviço de diagnóstico. Naturalmente, como cada serviço reage de uma forma diferente à monitoria, é necessário um processo interface para cada um dos serviços. No item 4.3 explicaremos detalhadamente como implementamos a monitoria de serviços através do uso de processos interface. Cada processo interface definido implementa a interface de monitoria do SDF.

Na figura 3.5 podem ser vistos três serviços distintos que fazem parte da aplicação distribuída e que são monitorados pelo SDF através de processos interface. O processo P1 representa o processo interface para monitoria de um Servidor WEB, o processo P3 representa o processo interface para a monitoria do Serviço de Correio e o P6 é o processo interface para monitorar um serviço JAVA. Os outros processo (P2, P4 e P5) são processos remotos da aplicação.

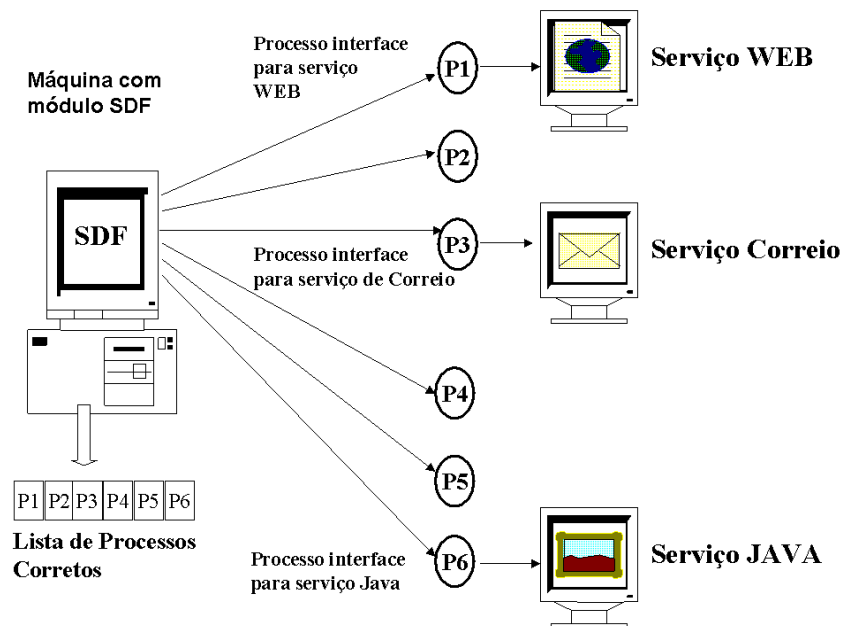


Figura 3.5 - Diferentes tipos de processos e serviços monitorados pelo SDF

Em cada módulo, na lista de processos corretos existem os processos remotos da aplicação e os processos interface que permitem a monitoria de serviços importantes para o funcionamento da aplicação distribuída.

3.5. Tipos de Diagnósticos de Falhas

O acompanhamento dos tempos de comunicação entre os módulos SDF e os processos monitorados permite a identificação de falhas que impactam na aplicação distribuída. Estes níveis de tempo de comunicação são definidos pelos usuários do SDF, no momento da inserção do processo para monitoria, e guardados em uma tabela do objeto gerente do submódulo de detecção de falhas. Para cada processo são adotados três níveis aceitáveis de valores de tempo de comunicação. Nada impede, no entanto, que uma quantidade maior de níveis possa ser definida a partir de ajustes nos parâmetros do SDF. Cada um destes níveis

de comunicação está relacionado com um nível de qualidade de serviço fornecido pela aplicação e compreende uma faixa (intervalo) de valores.

No momento em que o agente de conexão está monitorando um processo, ele compara o valor encontrado na comunicação (TC) com os valores definidos de tempo de comunicação guardados na tabela do objeto gerente. Os valores fornecidos pelo usuário (TC1, TC2 e TC3) para cada processo devem ser baseados em estimativas feitas previamente, que levem em consideração a tecnologia de rede e o comportamento da infraestrutura de comunicação. O primeiro nível de tempo de comunicação, TC1, corresponde à situação ideal de comunicação com o processo, TC2 a uma situação intermediária e TC3 corresponde a uma situação de comunicação degradada mas ainda aceitável para continuidade da aplicação distribuída. Se o valor encontrado em TC for maior do que o primeiro nível de tempo de comunicação (TC1) então o SDF compara com o segundo (TC2). Se esse também for maior, o SDF compara o valor encontrado com o terceiro nível (TC3). Para o SDF valores encontrados na comunicação com o processo até o terceiro nível de tempo de comunicação (TC3) significa que o processo está respondendo dentro de valores aceitáveis.

A implementação dos diferentes níveis de tempo de comunicação, *timeouts* adaptativos, segue a idéia proposta em [MR98, Mac00].

O SDF identifica diferentes situações de falhas, a saber: processo monitorado não responde dentro dos limites de tempo de comunicação especificados para ele; falha de um processo monitorado; falha na máquina onde residem processos monitorados; e falha do módulo SDF de uma máquina que tenha processos monitorados em estado suspeito. Nos itens a seguir, examinaremos cada uma destas situações de falhas.

3.5.1. Processo não Responde Dentro dos Limites de Tempo de Comunicação Especificados

Em algumas situações, o agente de conexão detecta que, apesar de conseguir contatar o processo monitorado, não está conseguindo fazê-lo dentro dos tempos definidos para a comunicação, isto é, até TC3. Neste caso, o processo é sinalizado como estando suspeito mas não é retirado da lista de processos corretos da aplicação. Isso se dá porque o problema

existente é na comunicação e não de falha do processo. No momento em que o processo voltar a responder dentro dos níveis de tempo de comunicação definidos para ele, este para de ser considerado suspeito. Neste caso, a sinalização de um processo como suspeito indica que, temporariamente, este apresenta problemas na comunicação.

Na figura 3.6 vemos o agente de conexão (AC) da máquina 1 monitorando os processos P1, P2 e P3. Os símbolos R, G, AC e AO representam respectivamente os objetos reconfiguração, gerente, agente de conexão e agente de objetos. O processo P1 e o processo P3 são processos interfaces, que monitoram os serviços WEB e Correio respectivamente, retornando o resultado para o SDF. No momento da inserção dos processos para monitoria, foram definidos três níveis de tempo de comunicação.

Na figura 3.6, é possível observar que para o processo P2 foi definido TC1 igual a 10ms, TC2 igual a 20ms e TC3 igual a 30ms. Nesta figura, ilustramos a situação onde, em um dado instante, o tempo de comunicação para atingir o processo P2 é de 45ms. Este valor está acima do especificado na tabela do objeto gerente, tornando o processo P2 suspeito.

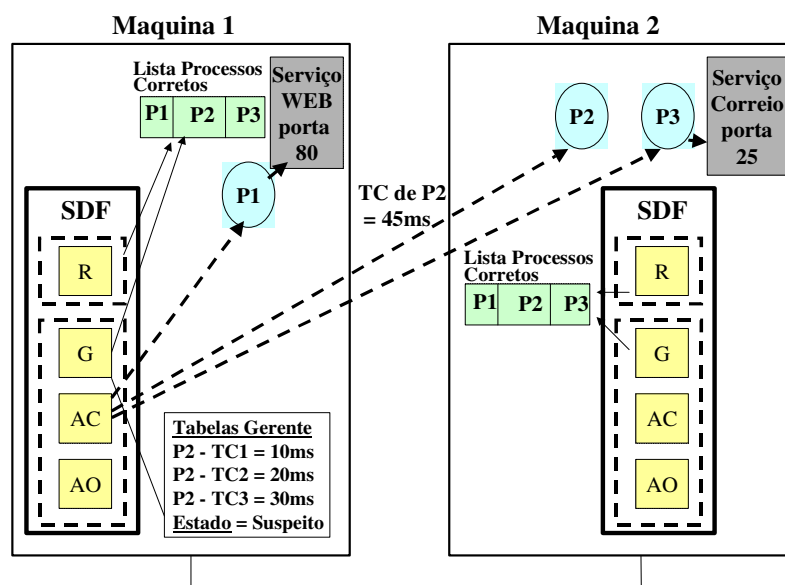


Figura 3.6 - Estouro de tempo de comunicação do Processo P2

Com a definição de níveis diferentes de tempo de comunicação, o SDF contorna⁴ situações de falhas originadas de instabilidade na comunicação com os processos. Ao mesmo tempo sinaliza para a aplicação a situação da infraestrutura de comunicação.

3.5.2. Processo não Está mais na Tabela do Sistema Operacional ou não Responde ao Módulo do SDF Local

Quando um agente de conexão, de uma das máquinas do grupo, percebe que um processo monitorado não responde, ele avisa ao gerente do módulo SDF local que houve uma falha. Se o processo que falhou for local, então o agente de objetos da máquina é acionado. Se o processo é remoto, então deve ser solicitado ao agente de objetos da máquina, onde o processo reside, que investigue esta situação. Este agente de objetos é o mais indicado para detectar a falha do processo, visto que as investigações serão locais, sem a interferência de fatores externos. Neste caso, não existem fatores como sobrecarga temporária da rede, perda de mensagens ou mesmo falha de máquina para interferir na investigação.

O agente de objetos faz investigações com a finalidade de verificar se o processo está ativo nas tabelas do sistema operacional. Quando é detectado que este está nas tabelas do sistema operacional, o agente de objetos testa-o através de chamadas locais. Um processo que não responde ao agente de objetos local, ou que não está nas tabelas do sistema operacional, não pode continuar na lista de processos corretos da aplicação. Um processo é denominado de moribundo se não responde a solicitações remotas/locais, mas ainda permanece nas tabelas do sistema operacional.

Na figura 3.7 vemos o agente de conexão da máquina 1 monitorando os processos da sua lista de processos corretos e suspeitando que existe uma falha em P3. Em seguida ele avisa ao gerente local que acionará o agente de objetos da máquina 2 (processo P3 é remoto). É possível ver, ainda, que ao receber a solicitação de investigação do gerente da máquina 1, o agente de objetos faz investigações no sistema operacional para determinar o estado do processo suspeito P3.

⁴ O SDF contorna situações de falhas (provinientes de instabilidade na infraestrutura) por tentar se comunicar com o processo com diferentes valores de tempo de comunicação (TC), antes de sinalizar uma falha.

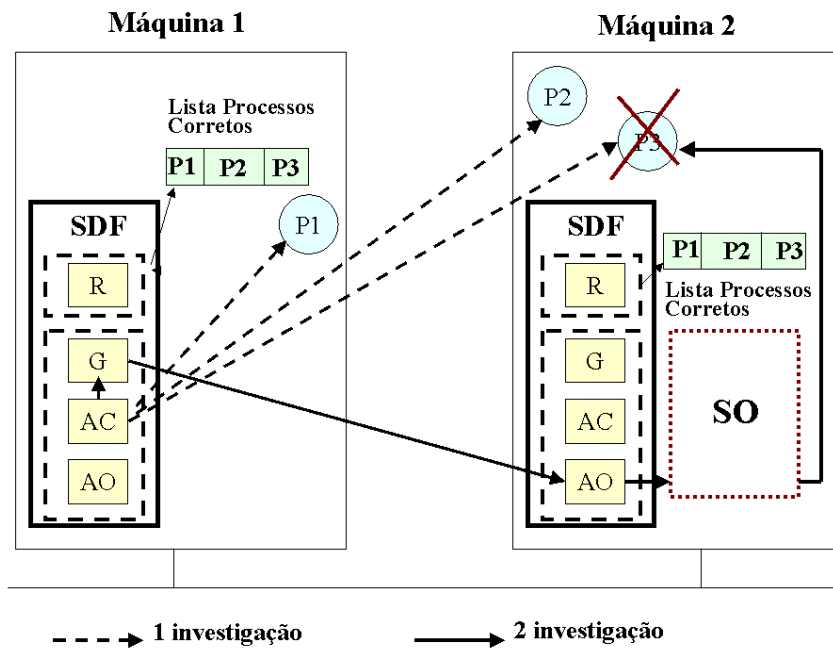


Figura 3.7 - Solicitação de investigação sobre processo no sistema operacional

Caso fique detectado pelo agente de objetos que o processo não está mais nas tabelas do sistema operacional, ou então, que apesar de estar não responde a chamadas locais, então o objeto reconfiguração é acionado para iniciar uma rodada de reconfiguração. Nesta reconfiguração todos os outros módulos SDF do grupo receberão um aviso para retirar o processo falho da sua lista de processos corretos. Neste caso, não há necessidade de acordo entre os módulos pois um processo que não responde nem ao módulo SDF existente na mesma máquina que ele, com certeza não está apto a fazer parte do grupo. Situações onde é necessário um acordo entre os módulos para retirada de processos serão detalhadas nos itens seguintes.

A figura 3.8 representa a situação descrita anteriormente onde o agente de objeto detecta que P3 não está nas tabelas do sistema operacional. Ele aciona o gerente do módulo local que, por sua vez, solicita ao objeto reconfiguração o início de uma rodada para retirar P3 das listas de processos corretos do grupo.

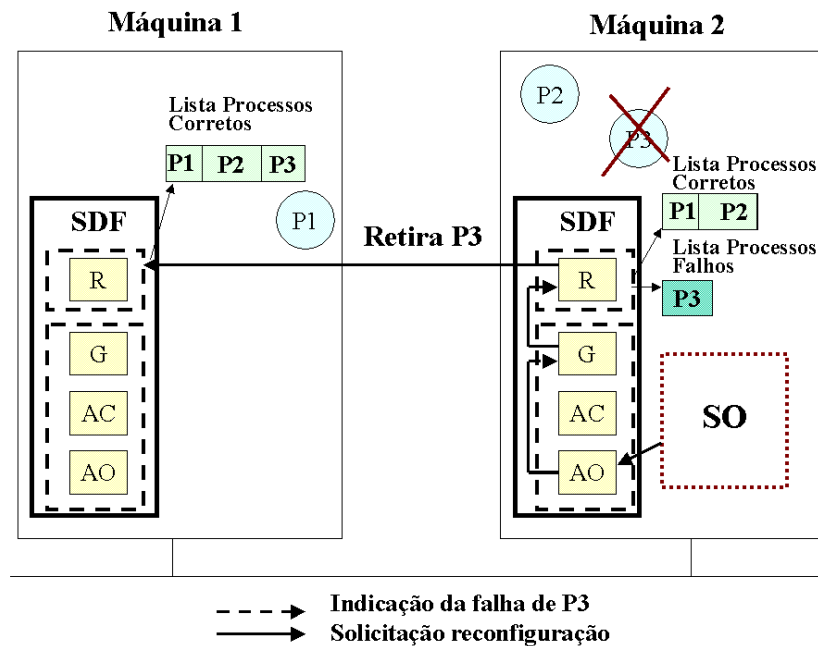


Figura 3.8 - Agente de objeto detecta falha do processo P3 e objeto reconfiguração inicia uma reconfiguração

Como todos os objetos agente de conexão do grupo investigam simultaneamente todos os processos, é possível que o agente de objeto da máquina do processo falho seja acionado por vários módulos SDF. Quando isso acontece o agente de objetos executa apenas a primeira solicitação avisando aos outros módulos que já existe uma investigação sobre o processo suspeito. Esta primeira solicitação pode ser proveniente de qualquer um dos módulos, dependendo exclusivamente da ordem de execução dos testes sobre os processos.

Com investigações na tabela do sistema operacional e testes locais, o SDF consegue identificar falhas do tipo *crash* em processos suspeitos. O módulo SDF que determina falhas desta natureza é aquele que reside na máquina do processo falho. Neste caso, não é necessário acordo entre os módulos para retirada do processo do grupo.

3.5.3. Falha na Máquina que Contém Processos Monitorados.

Os agentes de conexão suspeitam também de falha de máquinas com processos monitorados. Eles acionam a porta ECHO para verificar se a máquina responde a

solicitações remotas. O serviço TCP/IP ECHO no sistema operacional é utilizado para testes de conectividade. Em todas as máquinas ele é habilitado para ser utilizado pelo SDF a fim de determinar se uma máquina remota está respondendo às solicitações.

Quando um agente de conexão não consegue contactar uma máquina, esta é marcada como suspeita de falha e seus processos são marcados como suspeitos (observe que um particionamento físico da rede pode também resultar na suspeita da máquina). Para retirar os processos dessa máquina do grupo é necessário assegurar que os outros módulos do SDF também tenham detectado a falha. Para isso propomos um protocolo de acordo que é descrito no item 3.6.

A figura 3.9 ilustra o instante quando o agente de conexão da máquina 1 percebe que a máquina 2 e seus processos não podem ser contactados. A falha da máquina é identificada como o motivo da falha dos processos.

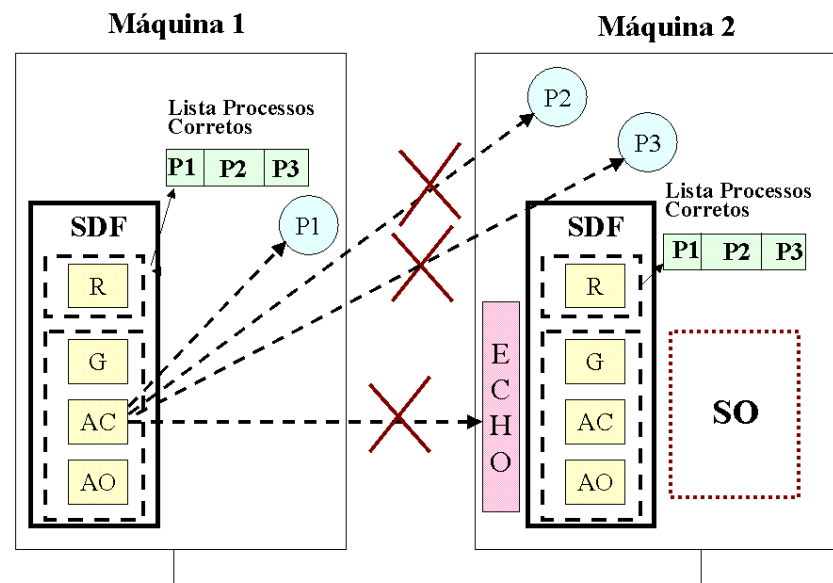


Figura 3.9 - O módulo SDF da máquina 1 não consegue contactar a máquina 2 e seus processos

Assumimos, no funcionamento do SDF, que uma máquina está falha se realmente tiver inativa ou se tiver sem contato com o servidor de nomes. Isto se dá, pois é possível que uma máquina falha para os módulos SDF do grupo, esteja, na verdade, sem contato com o grupo

e com o servidor de nomes. Neste caso, a própria máquina particionada sabe que seus processos locais serão retirados das listas de processos corretos do grupo e que as referências aos seus objetos serão apagadas do servidor de nomes, conforme determina o procedimento de retirada de uma máquina do grupo (item 3.6). Assim, ela se prepara para esta situação, colocando os seus processos locais em uma lista de excluídos.

No momento em que a comunicação com o servidor de nomes for restabelecida, as referências aos seus objetos serão inscritas novamente ali e seus processos locais retornam ao grupo para serem monitorados.

A figura 3.10 mostra uma máquina na situação de falha descrita. No exemplo da figura, foi uma falha no hub que levou a máquina 3 a ficar sem acesso ao servidor de nomes. É possível perceber que os outros módulos SDF executaram um protocolo de acordo e retiraram os processos da máquina 3 das suas listas de processos corretos, colocando-os nas listas de processos falhos. Como o problema não foi uma falha da máquina, a máquina 3, ao perceber sua atual condição, coloca seus processos locais em uma lista de excluídos e começará a monitorar o acesso ao servidor de nomes. Assim que esta máquina tiver novamente contato com o servidor de nomes, ela solicita o estado atual do grupo (suas listas de processos corretos e falhos). Solicita também que as referências a seus objetos sejam novamente inscritas no servidor de nomes e que os processos existentes na sua lista de excluídos sejam inseridos no grupo para monitoria.

Desta maneira, o SDF trata também situações de falhas temporárias de máquinas conforme visto na figura 3.10. Ele possibilita aos processos, de uma máquina sem contato com o servidor de nomes, retornar ao grupo depois que o problema na comunicação for resolvido.

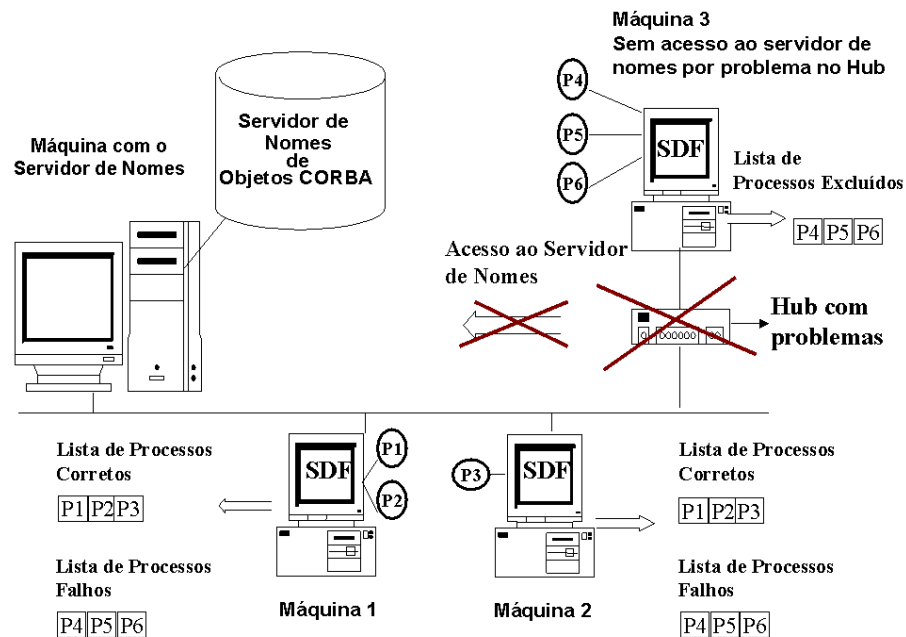


Figura 3.10 - Máquina sem acesso ao servidor de nomes e as ações decorrentes desta situação

3.5.4. Falha do Módulo SDF de uma Máquina que Tenha Processos Monitorados em Estado Suspeito.

Consideramos que a falha de um dos componentes do módulo SDF local implica na falha do módulo como todo. Este tipo de falha impacta na monitoria dos processos apenas quando um processo, vinculado a este módulo SDF⁵, se torna suspeito. Se um módulo SDF de uma máquina falhar, mas os processos desta máquina continuarem a funcionar corretamente, o serviço de diagnóstico distribuído não tratará desta falha até que algum dos processos da máquina se torne suspeito. Neste caso, o processo não responde à monitoria dos módulos SDF, a máquina onde ele reside responde corretamente às chamadas ao serviço Echo, mas o agente de objetos da máquina não pode ser contactado para investigar o estado do processo suspeito.

⁵ Um processo é vinculado ao módulo SDF que monitora a sua aplicação e que está na mesma máquina. O vínculo existe pois apenas este módulo pode determinar a falha individual do processo (conforme pode ser visto no item 3.5.2).

A figura 3.11 representa a situação descrita acima onde o módulo SDF da máquina 2 não pode ser contactado para investigar o estado de P3, conforme solicitação feita pelo gerente da máquina 1.

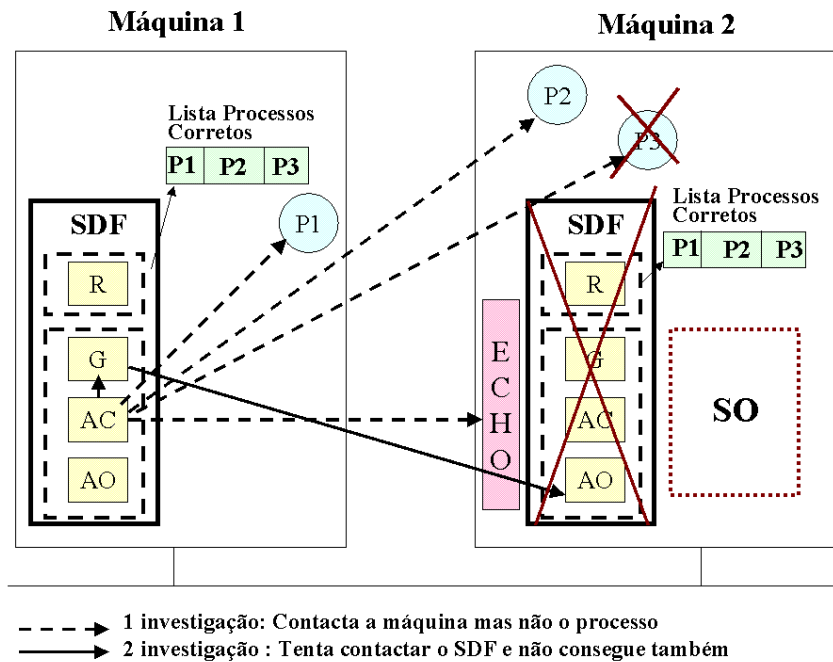


Figura 3.11 – Detecção de falha do módulo

Esta situação caracteriza-se como uma falha no módulo do serviço. Este tipo de falha impossibilita que investigações sobre o estado dos processos sejam feitas. Inicia-se então um protocolo de acordo entre os módulos SDF do grupo para validar a retirada (por falha no módulo SDF) dos processos desta máquina da lista de processos corretos.

Um módulo SDF que não tenha processos locais monitorados pelo grupo, pode tornar-se falho também. Isto é descoberto pelos outros módulos SDF no momento de um acordo sobre determinada máquina. Como não existem processos associados a ele, assim que é descoberta a sua falha as referências aos objetos do módulo são retiradas do servidor de nomes.

Com o diagnóstico de falhas de seus próprios módulos, o SDF apresenta-se como um serviço tolerante a falhas.

3.6. Protocolo de Acordo Utilizado no SDF

O protocolo de decisão em duas fases, utilizado no SDF, é tolerante à falha do coordenador e se baseia no protocolo *Two-Phase Commit* [CDK96].

O protocolo *Two-Phase Commit* é considerado um protocolo eficiente e “barato” por necessitar somente de três passos para comunicação e $3n$ mensagens para confirmação da transação (onde n é o número de participantes do protocolo). No entanto, ele não evita bloqueio ou seja, se o coordenador falhar enquanto alguns participantes estão em estado incerto (quanto ao resultado da transação), estes participantes terão de esperar pela volta do coordenador para receberem a decisão final da transação. Uma alternativa para resolver esta situação de bloqueio é obter, de algum outro participante, a decisão final sobre a transação. Todavia, isso não funciona corretamente se todos os participantes ativos estiverem em estado incerto.

Segundo [GLS96] um protocolo evita bloqueio se ele permite que a decisão (*commit* ou *abort*) seja atingida por participantes em estado correto independente das falhas de alguns. Bloqueios podem ser evitados por adicionar mais fases para confirmação da transação, como é feito no protocolo *Three-Phase Commit* [BHG87]. Neste protocolo, um participante não confirma a transação até saber que todos votaram pela sua confirmação. Desta forma, o protocolo *Three-Phase Commit* necessita de cinco passos para comunicação com $5n$ mensagens (dois passos de comunicação a mais e $2n$ a mais de mensagens do que o protocolo *Two-Phase Commit*). Por conseqüência, o *Three-Phase Commit* tem uma latência maior, isto é, precisa de mais tempo para confirmar uma transação. Outros protocolos que evitam bloqueio foram propostos em [GS95, GLS96, AGP98].

Por uma questão de eficiência (quantidade menor de mensagens no protocolo), optamos por basear o protocolo de acordo do SDF no protocolo *Two-Phase Commit*, com a implementação de tolerância a falha do coordenador.

Inicialmente, o coordenador do grupo é o objeto reconfiguração do primeiro módulo SDF inscrito no servidor de nomes. O coordenador lidera o acordo entre os módulos com a finalidade de retirar do grupo os processos de máquinas falhas ou de máquinas cujo módulo SDF tenha falhado. Quando há falha (ou temporário afastamento do grupo) de um

coordenador, os participantes ativos (módulos SDF sem falhas) elegem um novo coordenador para o grupo. Este retomará as transações pendentes assim que eleito. Desta forma, segundo a definição apresentada em [GLS96] de um protocolo que evita bloqueio, o protocolo de acordo do SDF pode ser considerado baseado no *Two-Phase Commit*, mas com característica não bloqueante.

Na primeira fase do protocolo *Two-Phase Commit* do SDF, o coordenador pergunta a todos os envolvidos se estão preparados para confirmar a transação e aguarda as respostas. Na segunda fase, se todos os envolvidos estiverem preparados para efetuar a transação, ou seja, tenham respondido sim à transação em questão, então o coordenador solicita que eles a efetivem (*commit*). Caso os envolvidos não estejam preparados para efetuar a transação então o coordenador solicita que esta seja anulada (*abort*).

A seguir, veremos detalhes deste protocolo, a forma como acontece a eleição de um novo coordenador e algumas considerações finais sobre seu funcionamento.

3.6.1. Fases do Protocolo *Two-Phase Commit* do SDF – Retirada de Máquinas ou Módulos Falhos

Na primeira fase do protocolo *Two-Phase Commit* do SDF, o coordenador, ao detectar a falha de uma máquina com processos monitorados, envia a todos os outros objetos reconfiguração uma solicitação de voto quanto ao estado da máquina em questão, esperando em seguida um tempo para a resposta. Paralelo a esta ação, o coordenador conta quantos módulos ativos existem no grupo. Ele faz isso através de chamadas aos objetos reconfiguração que retornam mensagens de “*I am alive*”.

Na segunda fase, o coordenador conta o número de votos recebidos, comparando com o número de módulos ativos. Se todos os objetos reconfiguração votaram e concordaram que a máquina não pode ser mais contactada, então o coordenador solicita a todos os módulos que retirem os processos desta máquina da lista de processos corretos, colocando-os na lista de processos falhos. Após o acordo, o objeto reconfiguração apaga a referência do módulo SDF daquela máquina do servidor de nomes.

Uma rodada similar de acordo acontece quando o processo que está sendo monitorado não pode ser contactado e o módulo do SDF também não está disponível, apesar da máquina estar ativa.

A figura 3.12 mostra a primeira fase do protocolo descrito acima e a figura 3.13 a segunda fase deste protocolo. Na figura 3.12 vemos a máquina 1 com o objeto reconfiguração coordenador solicitando os votos a todos os módulos SDF do grupo sobre o estado da máquina 3. Em seguida, vemos as máquinas enviando o seu voto sobre o estado da máquina 3 para o objeto coordenador na máquina 1.

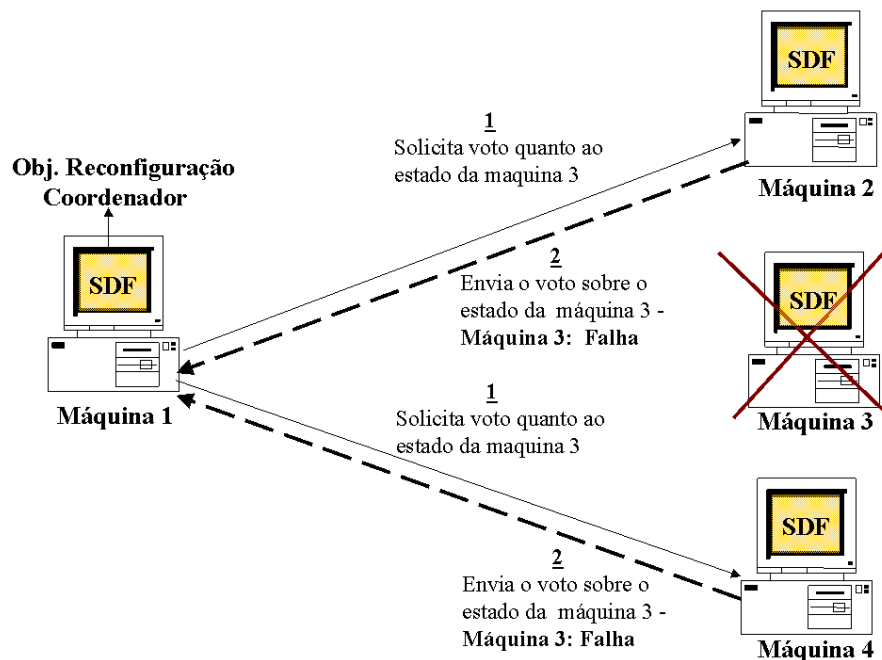


Figura 3.12 - 1º Fase do Protocolo de acordo para retirada dos processos de uma máquina falha

A figura 3.13 mostra as ações executadas após o acordo para retirada da máquina 3. A máquina 1, que contém o coordenador, solicita a todos os outros módulos SDF a retirada dos processos da máquina falha e, em seguida, apaga as referências do módulo SDF da máquina 3 do servidor de nomes.

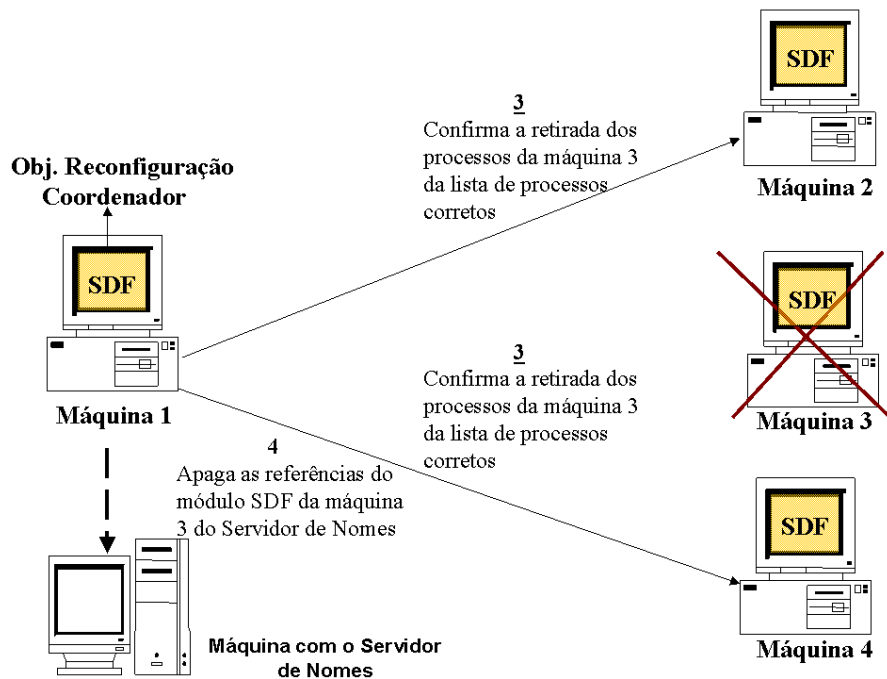


Figura 3.13 - 2º Fase do Protocolo de acordo para retirada de processos de uma máquina falha

3.6.2. Fases do Protocolo *Two-Phase Commit* do SDF – Eleição de um Novo Coordenador

No protocolo de decisão do SDF, se o objeto coordenador estiver em uma máquina que se torne falha, então um novo coordenador deve assumir com a concordância de todos os outros módulos SDF ativos.

Através de chamadas periódicas ao coordenador, os módulos percebem quando o coordenador falha. O objeto reconfiguração que tiver sua referência na próxima posição da lista do servidor de nomes, assumirá como candidato a coordenador.

Os outros objetos reconfiguração procuram, no servidor de nomes, o próximo objeto reconfiguração, testando-o para saber se este está ativo e guardando seu nome para o próximo coordenador (este teste é necessário para garantir que, em casos de múltiplas falhas de máquinas, o serviço de diagnóstico continue a operar normalmente).

Para validar a posição como coordenador, o candidato solicita a todos os outros módulos SDF a confirmação de que ele é o próximo objeto reconfiguração ativo da lista do servidor de nomes, e que deve ser o coordenador eleito. Se houver acordo para sua eleição (isto é, se o número de módulos SDF ativos for igual à quantidade de votos recebidos confirmando a eleição do coordenador), na segunda fase do protocolo, o novo coordenador do grupo avisa a todos que atualizem suas referências ao coordenador. Em seguida, os processos da máquina do antigo coordenador são retirados das listas de processos corretos do grupo e as referências aos objetos desta máquina são apagados do servidor de nomes.

Na eleição de um coordenador, o acordo acontece quando há unanimidade entre os módulos SDF ativos. Isto acontece, também, se apenas a máquina do novo coordenador estiver ativa.

Os módulos SDF falhos não são contados como ativos, nem seus votos são aguardados pelo candidato a coordenador. Se um módulo SDF falhar após ter sido considerado como ativo, o acordo só será estabelecido em uma eleição subsequente que acontecerá algum tempo depois de finalizada esta rodada de votos.

A figura 3.14 representa a primeira fase do protocolo de decisão para substituição do coordenador. Nesta figura, o coordenador do grupo reside na máquina 1, que falhou, e o próximo nome da lista do servidor de nomes é o objeto reconfiguração da máquina 3. Este candidato a coordenador solicita a todos os outros módulos que enviem a confirmação de falha da máquina 1 e de que ele é o próximo coordenador do grupo. Podemos ver também os módulos SDF enviando a confirmação para eleição do novo coordenador.

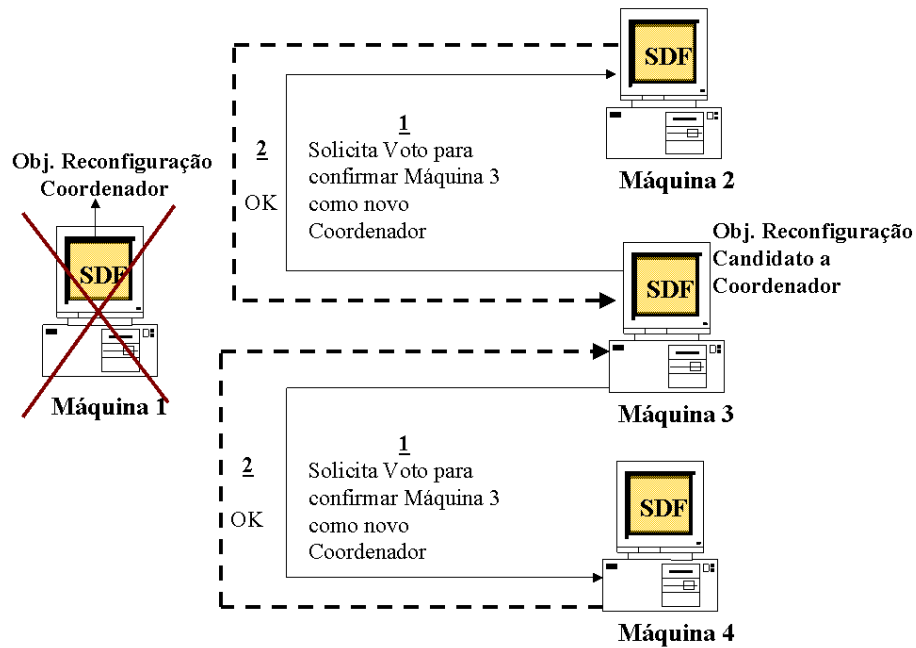


Figura 3.14 - 1º Fase do Protocolo de eleição do novo coordenador do grupo

A figura 3.15 apresenta a segunda fase do protocolo de eleição do coordenador. Neste caso, o objeto reconfiguração da máquina 3, após eleito coordenador, avisa a todos os módulos SDF que atualizem as referências ao coordenador e retirem da lista de processos corretos os processos da máquina 1. É possível ver também na figura o novo coordenador apagando as referências aos objetos da máquina 1 do servidor de nomes.

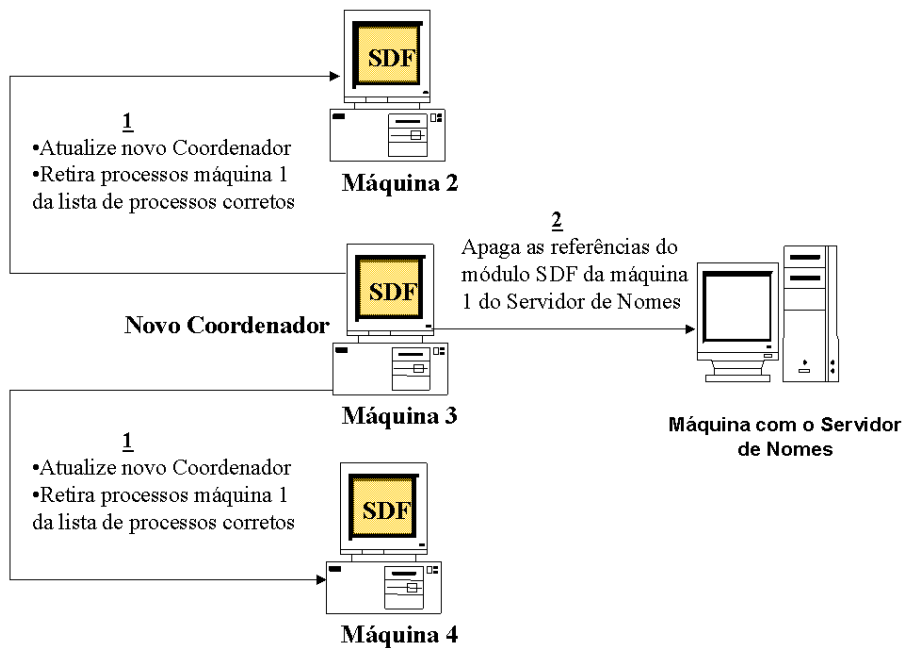


Figura 3.15 - 2ª Fase do Protocolo de eleição do novo coordenador do grupo

3.6.3. Considerações Sobre o Protocolo

O protocolo de decisão de duas fases, baseado no protocolo *Two-Phase-Commit*, tolerante a falha do coordenador, é um componente essencial para o serviço diagnóstico atingir aos objetivos propostos para ele. O SDF é tolerante a $N-1$ falhas de *crash* de seus módulos (onde N é a quantidade de módulos SDF inicializados em diferentes máquinas para uma dada aplicação). Ele também é tolerante a $M-1$ falhas de máquinas, onde M é o número de máquinas com módulos SDF. Ele pode funcionar com uma ou M máquinas, e é capaz de continuar a prover o serviço de acompanhamento de tempo de comunicação e detecção de falhas se até $M-1$ máquinas falharem simultaneamente. Isso é verdade também, se em uma destas máquinas estiver o coordenador do grupo. Em situações desta natureza, os módulos SDF restantes elegerão o próximo coordenador e em seguida o coordenador começará a tratar de cada uma das falhas das outras máquinas.

Por exemplo, considerando quatro máquinas no grupo, se três destas falharem simultaneamente (ou seus módulos SDF), o protocolo de acordo será executado pela máquina correta depois que esta se eleger coordenadora do grupo. Isto acontece pois na contagem dos módulos ativos apenas o módulo desta máquina será contabilizado e, conseqüentemente, apenas um voto será esperado para determinar o acordo.

Para evitar por completo inconsistências no grupo, o coordenador, de tempos em tempos, deve enviar a sua lista de processos corretos e falhos (com o motivo da falha) para que os outros módulos possam conferir seu estado no grupo. Módulos em estado igual ou mais adiantado que o coordenador não precisam fazer atualizações. Um estado mais adiantado significa ter recebido todas as informações de reconfiguração que ocorreram no grupo. Um estado atrasado em relação ao coordenador acontece quando o módulo tem algum processo como suspeito enquanto o coordenador já considera este como falho. Se o módulo estiver em um estado atrasado, então ao receber as listas do coordenador, ele identifica que houve um problema de reconfiguração e que, por isso, não recebeu a mensagem para transformar o processo suspeito em falho. Neste caso, o módulo deve se atualizar com as informações do coordenador.

3.7. Visualizador

O Visualizador é um componente da arquitetura do SDF que reporta, em uma interface gráfica, as informações de detecção e diagnóstico obtidas pelo serviço SDF. O uso do Visualizador permite informar o estado dos processos da aplicação e o histórico das máquinas do grupo. Um objeto Visualizador pode ser instanciado em qualquer máquina que tiver um módulo SDF funcionando. Suas informações mostram aos usuários os valores de tempo de comunicação com cada um dos processos corretos. Através do Visualizador, é possível ter dois tipos de visões: *acompanhamento dos processos* e *histórico das máquinas do grupo*.

A tela de *acompanhamento dos processos* apresenta todos os processos monitorados pelo serviço de diagnóstico, classificados pelo seu estado atual (corretos, falhos ou suspeitos).

Para os processos corretos é apresentado o valor atual de tempo de comunicação (TC), o nível que este valor se enquadra (TC1, TC2 ou TC3), e as informações complementares do processo como o ID, o endereço IP e o tipo do processo (processo comum da aplicação, processo WEB, processo Correio, processo Java, etc). Para os processos falhos, o Visualizador mostra informações similares às dos processos corretos acrescentando o tipo de falha ocorrida (processo fora da tabela do sistema operacional, falha na máquina onde reside o processo, falha no módulo SDF ou processo moribundo que não responde a chamadas locais apesar de ainda constar nas tabelas do sistema operacional). No caso de processos suspeitos, o Visualizador indica o motivo da suspeita sobre o processo (processo não responde dentro dos níveis de tempo de comunicação definidos para ele, processo não consegue ser contactado, máquina do processo não consegue ser contactada ou módulo SDF não responde a solicitações de investigação). Na tela de *acompanhamento dos processos* (figura 3.17), o botão “Informar Objeto” mostra o estado atual dos processos da aplicação cujo ID foi digitado. O botão “Parar Monitoria” finaliza temporariamente a monitoria realizada pelo Visualizador e o botão “Limpar Monitoria” limpa a tela.

A tela de *histórico das máquinas* apresenta o histórico das máquinas do grupo desde a sua criação até o momento atual. Informa máquinas que falharam e módulos SDF que foram finalizados em máquinas do grupo. Quando um módulo se inscreve no servidor de nomes, o Visualizador mostra, nesta tela, que a máquina do módulo está ativa no grupo. Quando uma máquina falha, a atividade de reconfiguração pode ser acompanhada através da tela de *histórico das máquinas*. O Visualizador apresenta o motivo pelo qual a máquina está sendo retirada do grupo e o acordo entre os participantes para confirmar esta retirada. Da mesma forma, quando uma máquina considerada falha pelo grupo restabelece o contato com o servidor de nomes, o Visualizador mostra que a máquina tornou-se ativa novamente. Neste caso, na tela de *acompanhamento dos processos*, os processos desta máquina, que estavam sendo considerados como falhos, são colocados como corretos, possibilitando o retorno da monitoria do SDF. Se algum destes processos tiver falhado durante a situação de particionamento ele é mantido na lista de processos falhos, no entanto o motivo da falha é modificado para indicar que esta ocorreu durante o particionamento da máquina. Todos os eventos são numerados a fim de mostrar a ordem dos acontecimentos no grupo. Na tela de

histórico das máquinas (figura 3.18), o botão “Informar Histórico” apresenta o histórico da aplicação cujo ID foi digitado e o botão “Limpar Histórico” limpa a tela do Visualizador.

Todas as informações disponibilizadas pelo Visualizador são retiradas de algumas tabelas do objeto gerente e do objeto reconfiguração da máquina onde o Visualizador foi instanciado. As informações que são mostradas nos Visualizadores existentes em uma rede são semelhantes, já que a atividade de reconfiguração envolve garantir uma visão coerente sobre os processos que estão sendo monitorados. Dizemos semelhantes pois a identificação dos processos falhos e do histórico das máquinas do grupo são iguais em todas as instâncias do Visualizador⁶ mas as informações apresentadas sobre os processos corretos dependem dos valores encontrados na monitoria destes. É possível, inclusive, que um processo seja considerado suspeito por um módulo (quando não responde dentro dos níveis de tempo de comunicação definidos entre eles) enquanto que, em outro módulo do grupo, este é considerado correto (o tempo de comunicação encontrado com o processo está dentro dos valores aceitáveis). Esta situação não indica uma incoerência de diagnóstico, uma vez que um processo considerado suspeito por não responder dentro dos valores definidos (TC1, TC2 e TC3), é tratado como um processo correto que não está atendendo temporariamente aos requisitos definidos pela aplicação. Neste caso, ele deve ser usado com ressalva ou suspeita. Na arquitetura do SDF, processos suspeitos por outros motivos aguardam um acordo do grupo para confirmar sua passagem para a lista de processos falhos.

As figuras 3.16, 3.17, 3.18, 3.19, 3.20, 3.21, 3.22, 3.23, 3.24 e 3.25 apresentam o Visualizador em variadas situações de detecção e diagnóstico, considerando um único grupo.

A Figura 3.16 mostra a tela inicial do Visualizador. A partir dela é possível chamar as telas *de acompanhamento dos processos* e a tela de *histórico das máquinas*. A tela de *acompanhamento dos processos* é chamada pela seleção da opção “Visão Geral” do menu Visões. A tela de *histórico das máquinas* é chamada através da opção “Histórico” do menu.

⁶ Visualizadores em máquinas que estão sem acesso ao servidor de nomes (temporariamente particionadas) apresentam uma mensagem informando que no momento não é possível mostrar o estado dos processos.

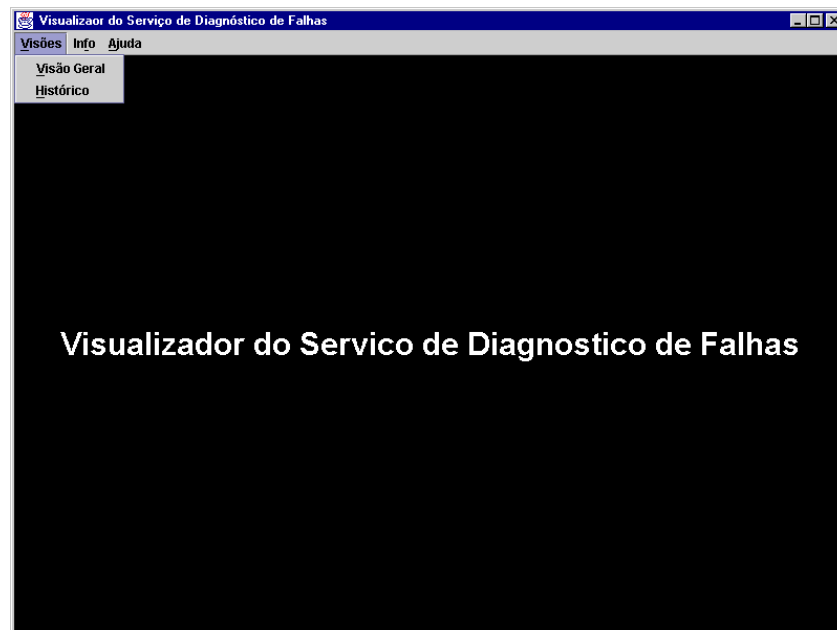


Figura 3.16 - Tela inicial do Visualizador

Na figura 3.17 podemos ver a tela de *acompanhamento dos processos* considerando um grupo inicial de três máquinas com módulos SDF (endereços IP 200.17.144.67, 200.17.144.73, 200.17.144.76) monitorando seis processos (ID's 111, 222, 333, 444, 555, 666). Os processos 111 e 222, que correspondem, respectivamente, a um processo interface para monitoria de um servidor WEB e a um processo aplicação, residem na máquina 200.17.144.76. Os processo 333 e 444 estão na máquina 200.17.144.73 e representam, respectivamente, um processo interface para monitoria de um serviço JAVA e um processo aplicação. A máquina 200.17.144.67 contém um processo interface para a monitoria do serviço de Correio (ID 555) e um processo aplicação (ID 666). Na figura 3.17 é possível observar que, exceto o processo 333, todos os outros estão respondendo dentro dos valores de tempo de comunicação estabelecidos em TC1.



Figura 3.17 - Tela de acompanhamento dos processos

A tela de *histórico das máquinas*, correspondente ao grupo descrito acima, pode ser vista na figura 3.18. Nela encontramos o registro da inicialização dos módulos SDF nas três máquinas, 200.17.144.76, 200.17.144.73, 200.17.144.67.

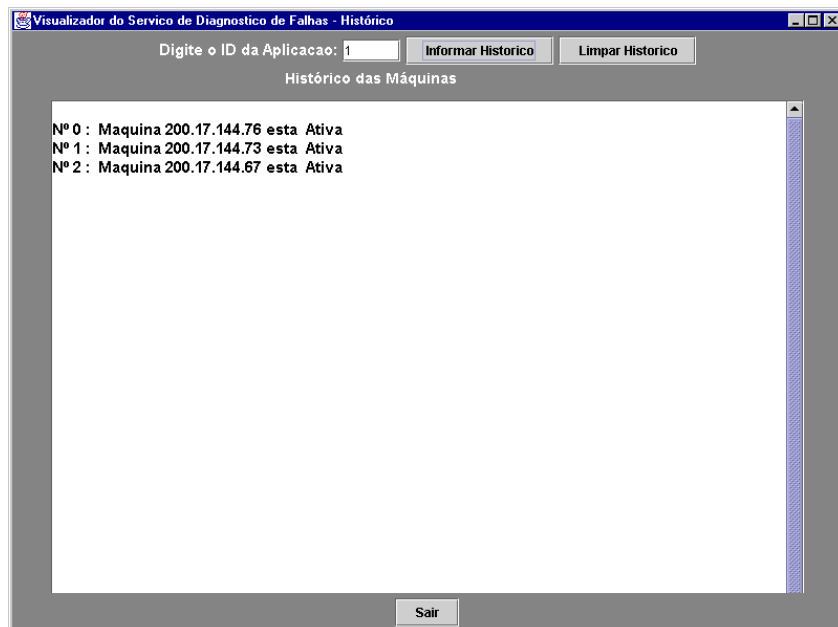


Figura 3.18 - Tela de histórico das máquinas

As figuras 3.19 e 3.20 refletem uma mudança ocorrida no grupo, dada à falha de uma das máquinas. A figura 3.19 mostra o momento em que os processos 333 e 444 são considerados suspeitos. O motivo apresentado é a falta de contato com a máquina onde eles residem, 200.17.144.73.

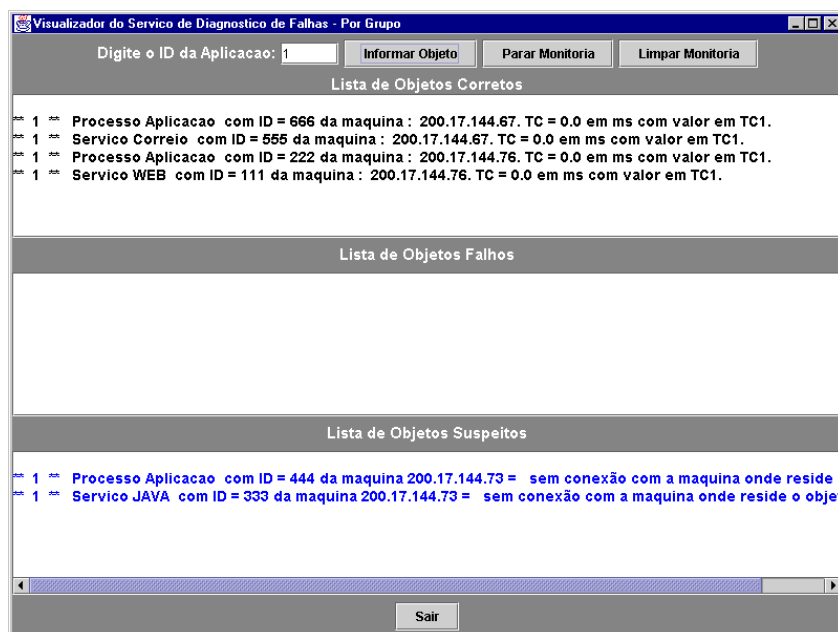


Figura 3.19 – Tela de acompanhamento dos processos – suspeita de falha nos processos 333 e 444

A figura 3.20 apresenta a detecção e o diagnóstico da falha do processo 333 e do processo 444. Estes processos são considerados falhos por sua máquina, 200.17.144.73, estar indisponível. Quando confirmada, com o acordo entre os módulos SDF ativos, a falha da máquina, no Visualizador estes processos são colocados na “Lista de Objetos Falhos”, como pode ser visto na figura 3.20. O motivo da falha dos processos, apresentados no Visualizador, é a falha na máquina onde estes residem.

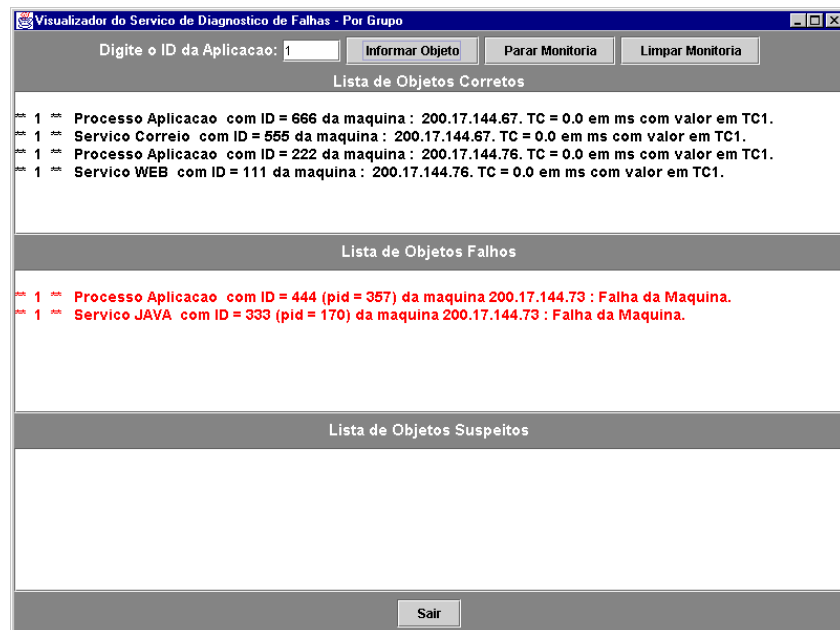


Figura 3.20 - Tela de acompanhamento dos processos com os processos 333 e 444 falhos.

A figura 3.21 apresenta a tela de *histórico do grupo*, com o registro da retirada da máquina 200.17.144.73 (que continha os processos 333 e 444). A informação “Nº 4”, desta tela, indica que houve acordo entre os módulos SDF para a retirada da máquina 200.17.1444.73.

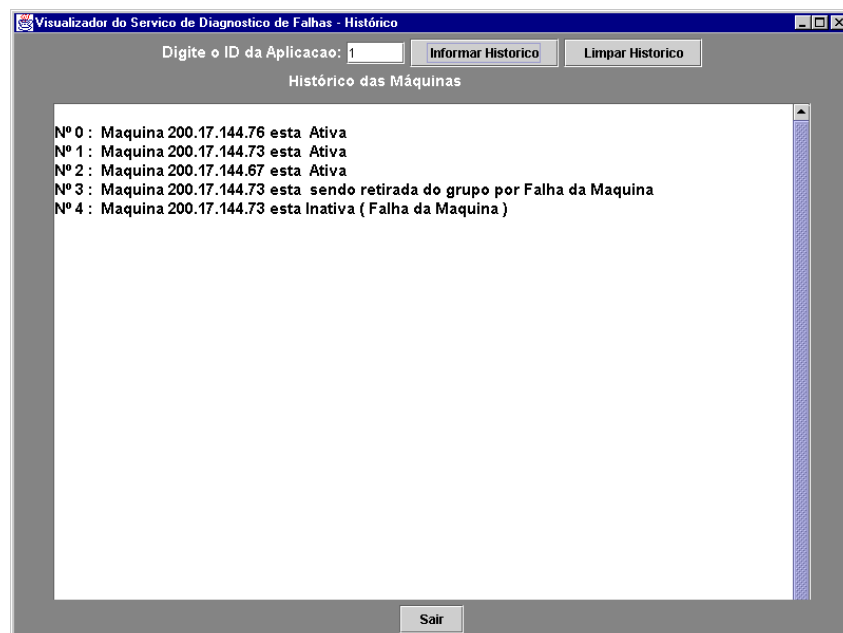


Figura 3.21 - Tela de histórico das máquinas com a retirada da máquina 200.17.144.73

Na figura 3.22 podemos ver na tela de *histórico* a volta da máquina 200.17.144.73 ao grupo. Esta volta se deve a resolução do problema de particionamento entre a máquina e o servidor de nomes. Uma vez resolvido este problema, os processos 333 e 444 são inseridos novamente para monitoria. Neste caso, a tela de *acompanhamento dos processos* do grupo volta a ser similar à mostrada na figura 3.17.

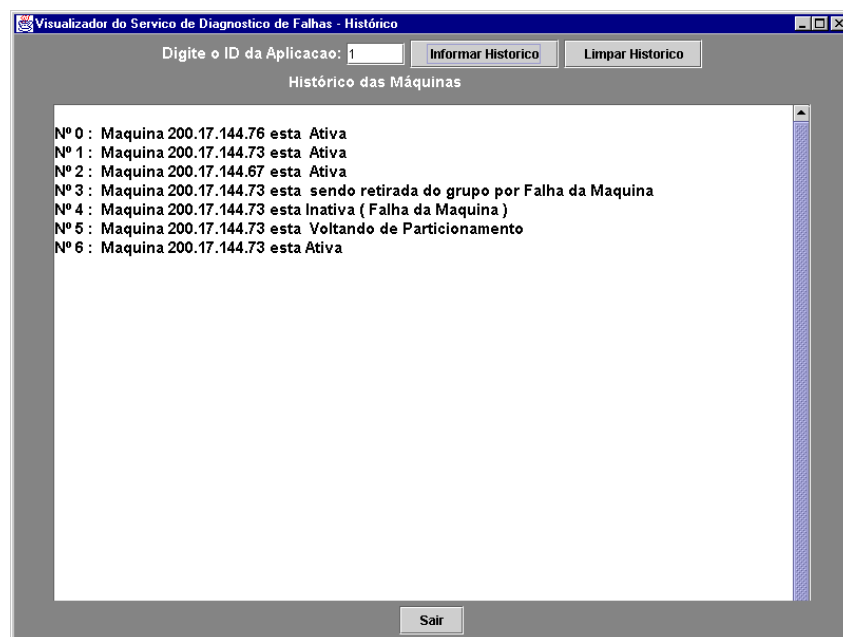


Figura 3.22- Tela de *histórico das máquinas* com o retorno da máquina 200.17.144.73

A figura 3.23 apresenta uma situação de falha individual de processo. O processo 111 foi retirado do grupo por não estar mais na tabela do sistema operacional da máquina 200.17.144.76, onde ele residia. A partir deste momento, ele aparece no Visualizador na “Lista de Objetos Falhos”. Esta falha, no entanto, não implica em mudanças na tela de *histórico das máquinas*.

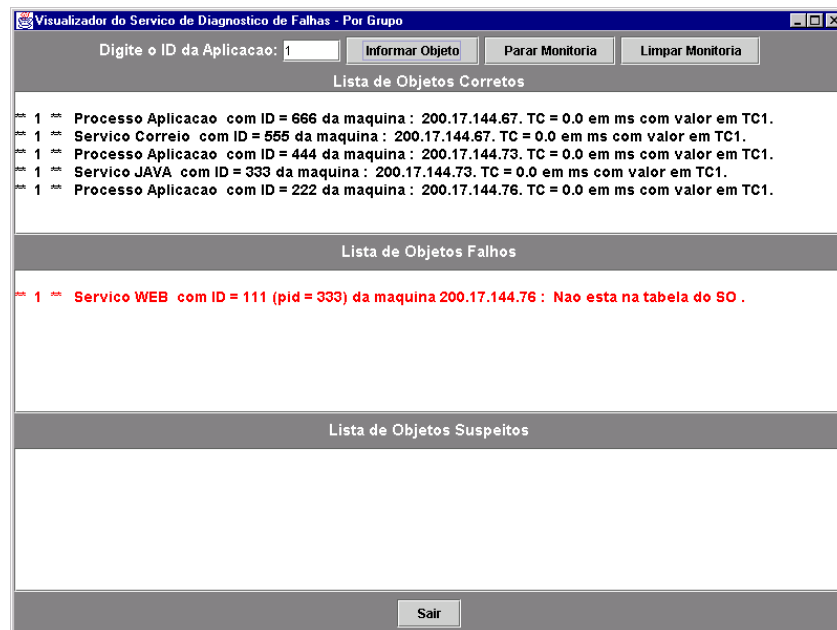


Figura 3.23 - Tela de *acompanhamento dos processos com falha do processo 111*

As figuras 3.24 e 3.25 mostram as mudanças ocorridas nas telas, *de histórico das máquinas e de acompanhamento dos processos*, quando um processo falha e o módulo SDF da sua máquina está falho. Nesta situação, como visto anteriormente, um acordo é necessário para retirar do grupo os processos vinculados ao módulo SDF que falhou. A figura 3.24 apresenta a tela de *histórico das máquinas* com o resultado do acordo para retirada da máquina 200.17.144.67 por seu módulo SDF estar indisponível.

No caso de falha do módulo SDF não é possível determinar com precisão o tipo de falha que ocorreu com o processo suspeito. Assim, todos os processos vinculados ao módulo SDF falho são retirados do grupo e o motivo apresentado para isso é a falha do seu módulo, como pode ser visto na figura 3.25.

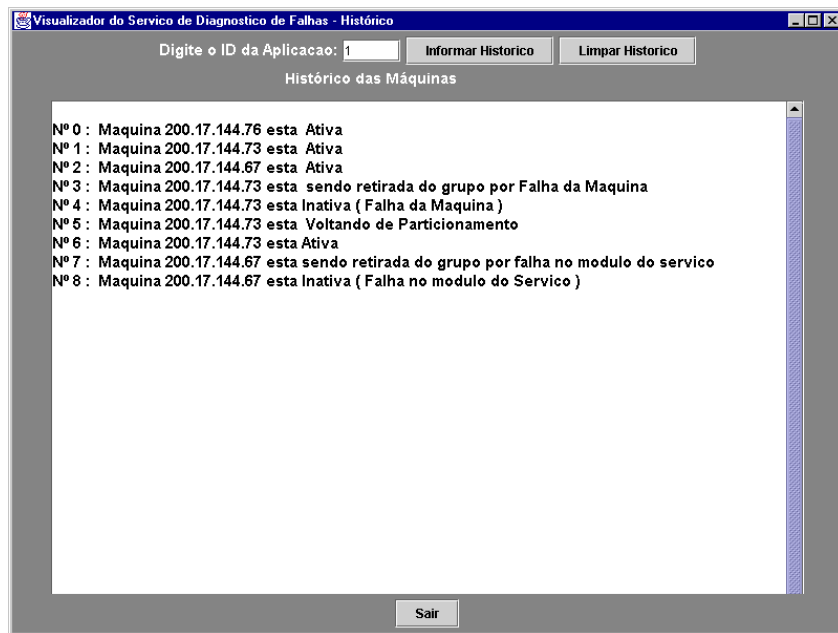


Figura 3.24 - Tela de *histórico das máquinas* - retirada da máquina 200.17.144.67 por falha no módulo

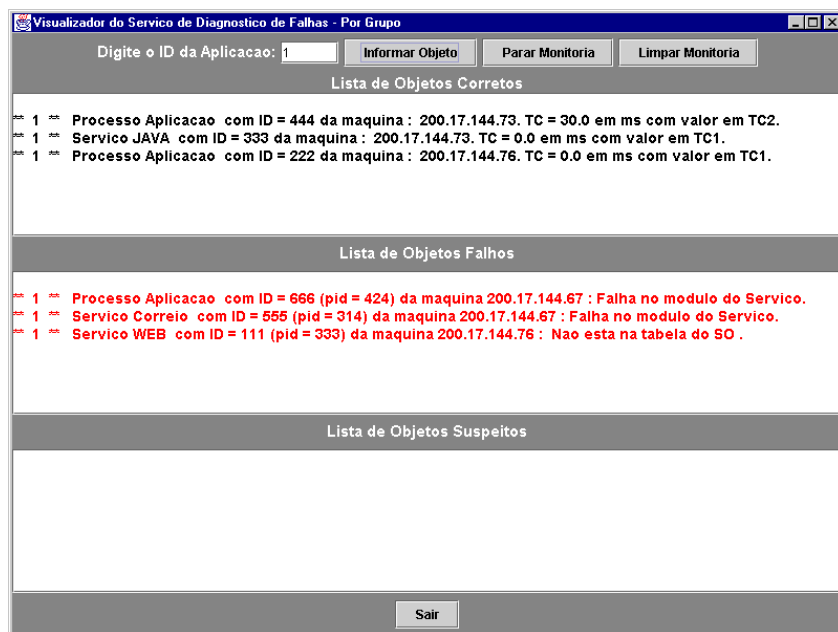


Figura 3.25 - Tela de *acompanhamento dos processos* - falha do módulo SDF da máquina 200.17.144.67.

Apresentamos, neste item, apenas algumas das situações de detecção e diagnóstico que podem ser acompanhadas pelo Visualizador. As outras situações representam variações destas e por isso não nos preocupamos em apresentar suas telas.

3.8. O SDF Estendido por Questões de Escalabilidade

Na versão apresentada e implementada do SDF, cada módulo monitora, de forma independente, cada processo que estiver na sua lista de processos corretos. Esta monitoria contínua de todos os processos do grupo tem como objetivo saber o valor atual do tempo de comunicação do módulo para cada processo, ao mesmo tempo que detecta as falhas de processos e máquinas do grupo.

Na figura 3.26 vemos este tipo de monitoria com seis processos distribuídos em três máquinas com módulos SDF. Neste caso, são necessárias dezoito mensagens de testes por rodada de monitoria (cada máquina monitora cada processo) para que todos os módulos tenham à disposição o tempo de comunicação com os processos e a detecção dos eventos de falhas.

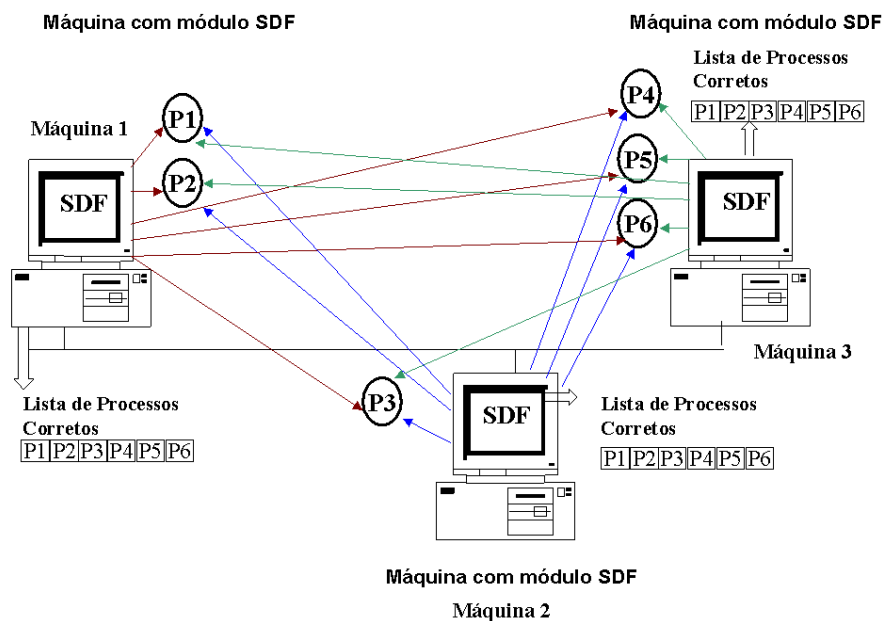


Figura 3.26 - Monitoria dos processos que estão na lista de processos corretos

Com este tipo de monitoria é possível ter um nível razoável de precisão quanto ao valor atual de tempo de comunicação com cada um dos processos monitorados, já que os testes

de conectividade são constantes. Esta é uma informação importante para um conjunto de aplicações⁷ que precisem rapidamente saber como está a comunicação com cada processo.

Ao mesmo tempo, para ter valores precisos de tempo de comunicação, que representem mais fielmente a situação da rede, é necessário um alto número de mensagens de testes.

No caso ilustrado na figura 3.26, cada módulo SDF, periodicamente, executa seis investigações para saber o estado dos processos de sua lista de processos corretos e seus tempos de comunicação. Com o acréscimo de mais processos, a quantidade de mensagens de testes na rede aumenta na proporção do número de módulos. Este tipo de solução, apesar de possibilitar a informação precisa do tempo de comunicação com o processo e a detecção mais rápida de falhas, pode apresentar problemas de escalabilidade quando existe um alto número de processos e um número expressivo de módulos SDF.

Para aplicações complexas, a alternativa proposta na arquitetura, mas não implementada na versão atual do SDF, é a adoção de uma monitoria representativa de processos e rotativa sobre as máquinas. Neste tipo de monitoria, periodicamente, cada módulo monitora os processos de uma outra máquina, afim de obter os valores de tempo de comunicação, ao passo que monitora um processo representativo por máquina para perceber quando acontece uma falha de máquina. Na próxima rodada o módulo monitora os processos representativos e os processos de uma máquina diferente da anterior, e assim sucessivamente.

A figura 3.27 mostra como seria este tipo de monitoria considerando o mesmo exemplo da figura 3.26. Nesta figura, é representada a rodada de testes onde, a máquina 1 monitora os processos da máquina 3, a máquina 2 monitora os processos da máquina 1, e a máquina 3 monitora os processos da máquina 2. Na próxima rodada teremos a máquina 1 monitorando os processos da máquina 2, a máquina 2 monitorando os processos da máquina 3 e a máquina 3 monitorando os processos da máquina 1. Além desta monitoria rotativa, tem-se, simultaneamente, uma monitoria representativa onde um processo é monitorado para indicar o estado da máquina. No exemplo da figura 3.27, o processo representativo da máquina 1 é o P1, da máquina 2 é o P3, e da máquina 3 é o P4.

⁷ Um exemplo deste tipo de aplicação é um balanceador de carga para servidores WEB. O balanceador precisa saber em quanto tempo os servidores estão respondendo as solicitações com a finalidade de determinar que servidor, no momento, tem melhores condições para atender requisições de clientes.

Se um processo representativo falha, um outro é utilizado para os testes. Com esta monitoria representativa de processos e rotativa sobre as máquinas teremos em cada rodada de monitoria, considerando o mesmo exemplo da figura 3.26, nove mensagens de testes (*números de módulos * quantidade representativa de processos – um por módulo*) ao invés das dezoito mensagens da monitoria completa (*números de módulos x quantidade total de processos*).

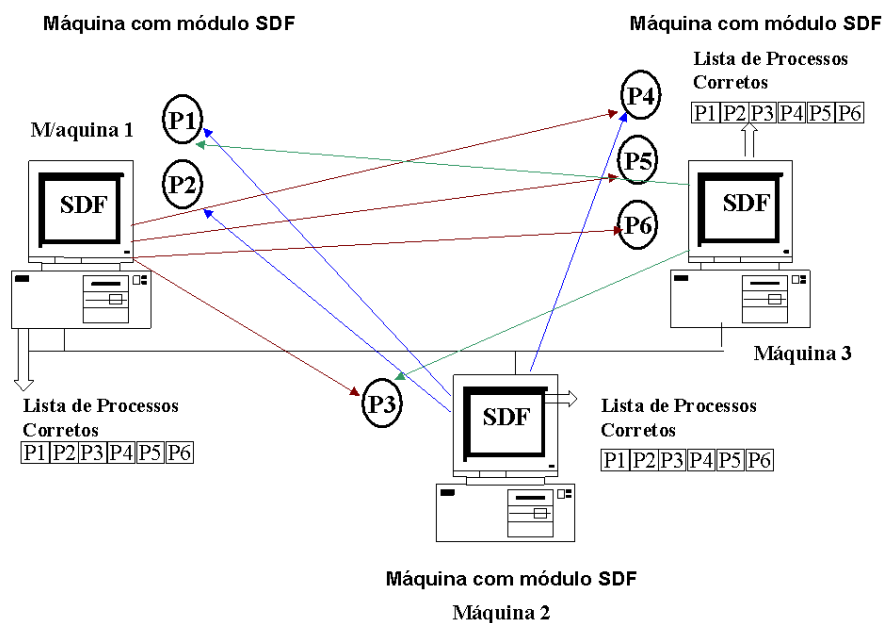


Figura 3.27 – Monitoria representativa de processos e rotativa sobre as máquinas do grupo

Com este tipo de monitoria perde-se em precisão quanto aos níveis de tempo de comunicação (já que os intervalos para investigações são maiores), mas diminui-se a quantidade de mensagens de testes na rede. Em situações de um alto número de máquinas no grupo e muitos processos monitorados, esta é uma alternativa da arquitetura que visa a escalabilidade do sistema.

Uma outra alternativa é utilizar o conceito de domínios de tolerância a falhas introduzido pelo padrão FT CORBA [OMG00], que será apresentado no capítulo 6 desta dissertação. Domínios de tolerância a falhas são definidos e a monitoria dos processos acontece dentro do domínio. Domínios diferentes trocam informação de detecção e diagnóstico através de

seus coordenadores, ficando o acompanhamento de tempo de comunicação restritos aos módulos SDF do domínio, como pode ser visto na figura 3.28. Nesta figura é possível identificar o domínio 1, com as máquinas 1, 2 e 3 e o domínio 2, com as máquinas 4, 5 e 6. Dentro destes domínios há monitoria dos processos para acompanhar os tempos de comunicação e identificar falhas. A troca de informação de detecção e diagnóstico entre domínios acontece através de seus coordenadores. Eles ficam responsáveis por repassar esta informação aos módulos do seu domínio de tolerância a falhas. O grupo de coordenadores também tem um coordenador dos domínios. Os protocolos para manter este grupo funcionam de forma similar aos apresentados anteriormente.

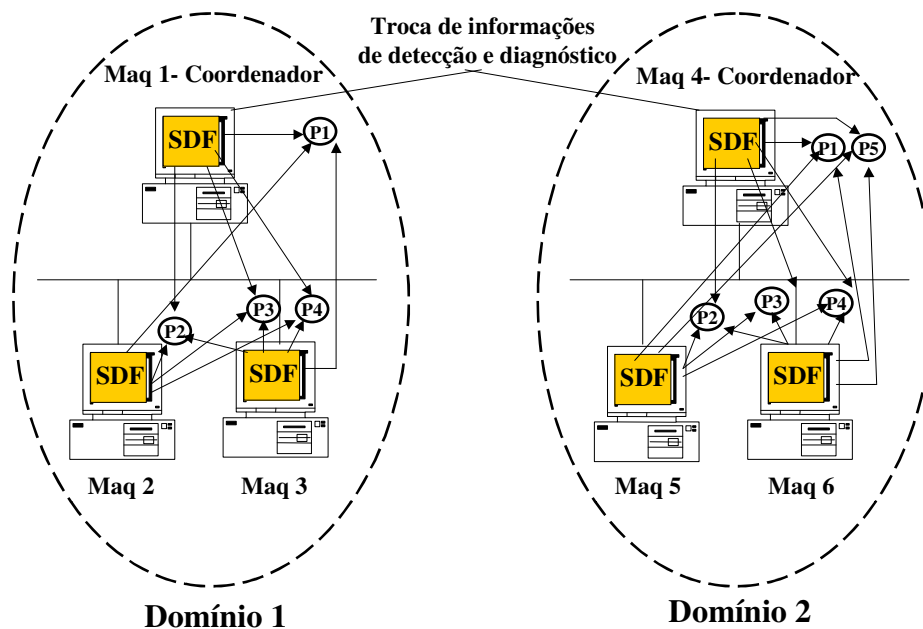


Figura 3.28 – O SDF em domínios de tolerância a falhas

3.9. Resumo do Serviço Apresentado

Neste capítulo apresentamos o serviço de diagnóstico de falhas e sua arquitetura. O SDF é um serviço que funciona sobre plataforma CORBA e que tem por objetivo detectar e diagnosticar processos de uma aplicação distribuída. Baseia-se nos conceitos de gerenciamento e diagnóstico visto no capítulo anterior. Utiliza-se de técnicas de detecção,

tais como a proposta em [Mac00], para identificar de forma mais precisa a ocorrência de falhas em processos. Identifica o tipo de falha, tomando as ações para manter as informações de diagnóstico coerentes e disponíveis para a aplicação de forma que esta possa tomar as ações desejadas. Estouro dos níveis de tempo de comunicação, falhas de processo, falhas de máquinas, falhas de módulos SDF são tratadas pelo serviço de diagnóstico. Um protocolo de acordo para retirada de uma máquina falha, ou com módulo SDF indisponível, também foi apresentado. Além da detecção de falhas, o SDF faz um acompanhamento do tempo de comunicação com os processos monitorados. Isto permite que a aplicação se adapte a diferentes faixas de qualidade de serviço.

O serviço de diagnóstico executa gerenciamento na medida em que informa o estado dos processos da aplicação distribuída e fornece a infraestrutura para que aplicações de gerenciamento sejam construídas. Ele é composto por módulos espalhados na rede que trabalham de forma conjunta. Cada módulo SDF é constituído pelo sub-módulo de reconfiguração e o sub-módulo de detecção de falhas. Os vários módulos SDF se comunicam através de um servidor de nomes CORBA.

Além de processos remotos da aplicação, o SDF monitora também serviços que não implementam a interface de monitoria do SDF. Ele faz isso através do uso de processos interface. Diferentes processos interface são propostos na arquitetura.

Apresentamos o Visualizador que é a ferramenta gráfica para reportar as informações de detecção e diagnóstico obtidas pelo SDF. Nas telas do Visualizador é possível acompanhar o estado dos processos da aplicação com seus valores de tempo de comunicação, bem como o histórico do grupo. Apresentamos, também, neste capítulo, as alternativas previstas na arquitetura do SDF para monitoria de aplicações complexas que necessitam de escalabilidade.

No próximo capítulo veremos como o serviço de diagnóstico foi implementado em Java sobre uma plataforma CORBA. Detalharemos suas classes e mostraremos alguns diagramas de classes e de seqüência.

Capítulo 4

Detalhes de Implementação do Serviço de Diagnóstico de Falhas

Neste capítulo apresentaremos a implementação do Serviço de Diagnóstico de Falhas em Java. Mostraremos as classes desenvolvidas e alguns diagramas de classes e seqüência.

4. Detalhes de Implementação do Serviço de Diagnóstico de Falhas

O ambiente para desenvolvimento utilizado na construção do SDF foi o JDK (*Java Development Kit*) versão 1.2.2, Visibroker (plataforma CORBA da Borland) para Java versão 3.4 e Visual C++ 4.0. O serviço de diagnóstico foi implementado como um conjunto de classes, desenvolvidas em Java, com operações necessárias para realizar a detecção e o diagnóstico de processos que estão sendo monitorados. Neste capítulo veremos os detalhes de implementação da arquitetura apresentada no capítulo 3.

As classes do serviço serão descritas resumidamente nos itens 4.1 e 4.2 desta dissertação. No item 4.3 serão mostradas as classes dos tipos de processos que podem ser monitorados pelo SDF e no item 4.4 as classes do Visualizador. Apresentaremos, no item 4.5, alguns diagramas de seqüência para modelar a interação entre os objetos do SDF. O objetivo é mostrar a seqüência cronológica das atividades realizadas pelo serviço durante a detecção de falhas e o diagnóstico. No item 4.6 mostraremos como estão organizadas as referências dos objetos dos módulos no servidor de nomes CORBA. É através destas referências que os módulos SDF em máquinas distintas se comunicam para realizar suas atividades de detecção e diagnóstico. No item 4.7 apresentaremos a estrutura de comunicação da plataforma CORBA e como esta foi utilizada para prover suporte à comunicação entre os objetos dos módulos SDF. Por último, faremos um resumo do capítulo, salientando os aspectos principais no item 4.8.

4.1. Classes que Compõem o Núcleo Básico do SDF

O SDF é inicializado em uma máquina pela execução da classe *ServidorObjetos*. Esta classe instância objetos que atuarão executando os papéis definidos na arquitetura para os objetos *reconfiguração*, *gerente*, *agente de objetos* e *agente de conexão*.

A figura 4.1 apresenta a relação entre os objetos propostos na arquitetura do serviço (visto na figura 3.3. do capítulo 3) e as classes que foram desenvolvidas para implementá-los. Conforme pode ser visto nesta figura, o objeto *reconfiguração* é implementado pela classe *ServicoReconfiguracao* e o objeto *gerente* através da classe *ServicoDFGerente*, onde a expressão “DFGerente” significa o gerente do módulo de detecção de falhas(DF). Por sua vez o *agente de objetos* e o *agente de conexão* são implementados pelas classes *ServicoDFAgenteObjetos* e *ServicoDFAgenteConexao*, respectivamente. Segundo a especificação da arquitetura, o objeto reconfiguração tem uma *lista de processos falhos* e *uma lista de processos corretos*. A *lista de processos corretos* é compartilhada também com os objetos *gerente* e objeto *agente de conexão*. Na nossa implementação do SDF, estas listas de processos foram construídas através das classes *ListaProcessosCorretosAplicacao* e *ListaProcessosFalhosAplicacao*.

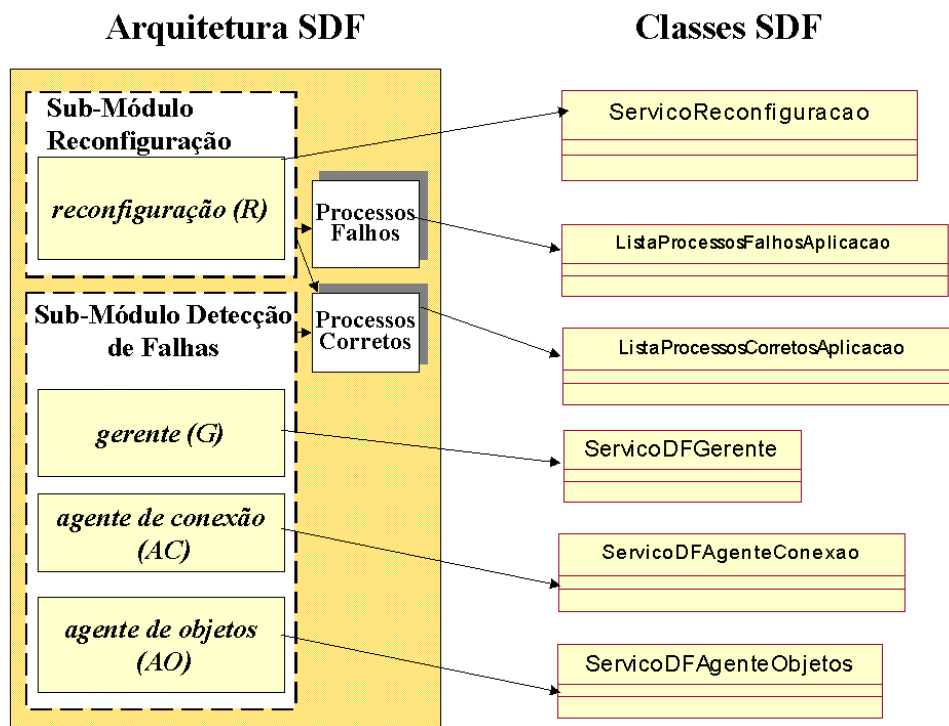


Figura 4.1 - Relacionamento entre a arquitetura proposta e as classes criadas

A figura 4.2 mostra um diagrama reduzido que apresenta as classes que fazem parte do núcleo básico do SDF com seus relacionamentos. Em cada módulo SDF que é colocado em funcionamento são instanciados: um objeto da classe ServicoReconfiguracao, um objeto ServicoDFGerente, um ServicoDFAgenteObjetos e um objeto ServicoDFAgenteConexao. O objeto ServicoReconfiguracao tem como atributos objetos das classes ListaProcessosFalhosAplicacao e ListasProcessosCorretosAplicacao. O objeto ListaProcessosCorretosAplicacao é compartilhado com os objetos ServicoDFGerente e ServicoDFAgenteConexao, como pode ser visto pelos relacionamentos existentes na figura 4.2.

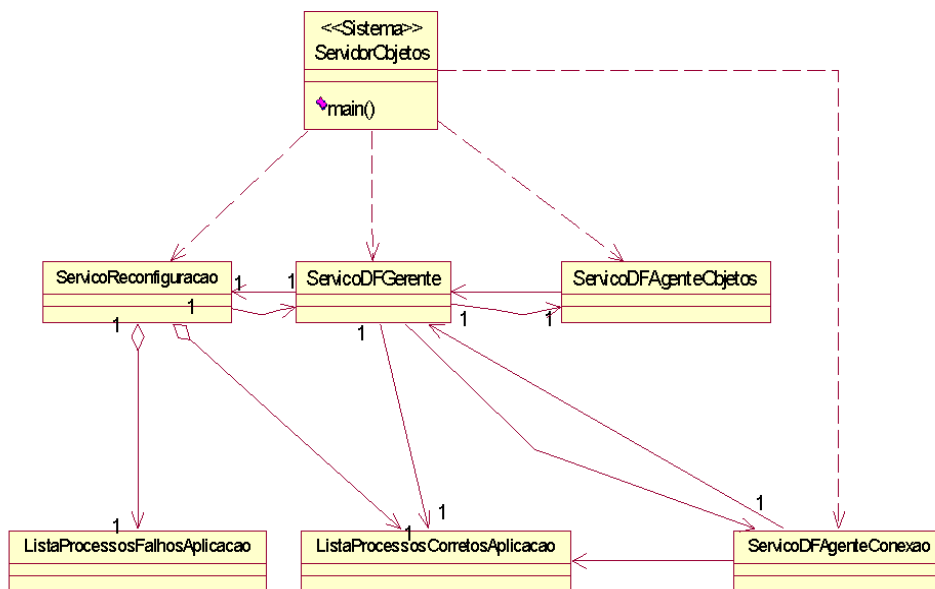


Figura 4.2– Diagrama de Classes do núcleo principal do SDF

A seguir apresentaremos cada uma das classes da figura 4.2 que compõem o núcleo básico do serviço de diagnóstico de falhas. As principais operações destas classes serão descritas de forma resumida, concentrando-se nas suas funcionalidades, sem que os parâmetros e valores de retorno sejam destacados.

4.1.1. Classe ServicoReconfiguracao

A classe ServicoReconfiguracao é responsável por manter uma visão coerente entre os módulos quanto ao estado dos processos monitorados. O conhecimento dos processos falhos e corretos da aplicação é responsabilidade dos objetos desta classe. Eles executam o protocolo de acordo, descrito no capítulo 3, item 3.6, entre os módulos para retirar do grupo os processos que residem em uma máquina falha, ou que estejam em um máquina com módulo SDF falho. Seus objetos também têm como atribuição atualizar as informações referentes ao histórico das máquinas onde existem módulos SDF e controlar a entrada de novos membros no grupo.

O primeiro objeto ServicoReconfiguracao que for cadastrado no servidor de nomes é denominado de coordenador do grupo. Como visto no item 3.6, o coordenador lidera o acordo para a confirmação de falhas de máquinas. Quando ocorre falha no coordenador, os outros objetos desta classe nos módulos remotos realizam entre si um protocolo de eleição de novo coordenador.

A seguir destacaremos alguns métodos da classe ServicoReconfiguracao que são invocados utilizando a plataforma CORBA. Estes métodos são operações executadas entre objetos da classe ServicoReconfiguracao que estão em máquinas distintas.

Na figura 4.3 temos a classe ServicoReconfiguracao com as operações executadas sob plataforma CORBA e em seguida teremos a descrição das operações listadas na figura.

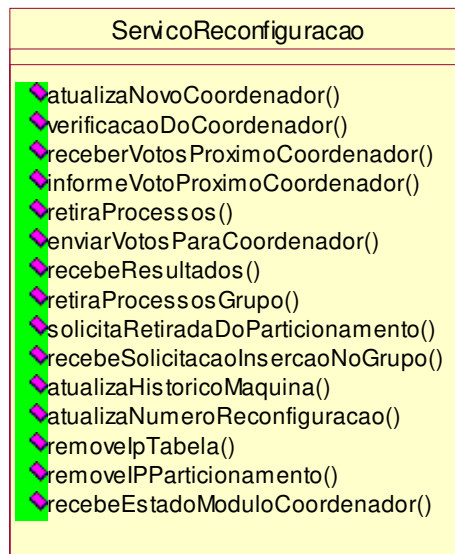


Figura 4.3 - Classe ServicoReconfiguracao - métodos utilizados pelos objetos da própria classe

A operação *atualizaNovoCoordenador()*

Esta operação é responsável por atualizar a informação referente a um novo coordenador. O solicitante é o coordenador que foi eleito e esta acontece sobre todos os objetos ServicoReconfiguracao existentes no servidor de nomes.

A operação *verificacaoDoCoordenador()*

A operação `verificacaoDoCoordenador()` tem por objetivo certificar que o objeto ServicoReconfiguracao coordenador do grupo está ativo. Ela é executada periodicamente pelos objetos ServicoReconfiguracao do grupo.

A operação *informeVotoProximoCoordenador()*

Operação que é solicitada a todos os objetos desta classe quando há necessidade de eleger um novo coordenador. O objeto solicitante é o candidato a coordenador.

A operação *receberVotosProximoCoordenador()*

Esta operação é executada por objetos *ServicoReconfiguracao* sobre o candidato a coordenador para que este receba os votos necessários à sua eleição.

A operação *retiraProcessos()*

Operação executada sobre todos os objetos desta classe existentes na rede quando há uma falha da máquina.

A operação *enviarVotosParaCoordenador()*

Esta operação solicita a todos os objetos *ServicoReconfiguracao* na rede que enviem seus votos quanto ao estado de uma determinada máquina que contém processos monitorados.

A operação *recebeResultado()*

Esta operação é executada sobre o coordenador do grupo, pelos objetos *ServicoReconfiguracao*, para informar seu voto sobre o estado de uma máquina suspeita.

A operação *retiraProcessoGrupo()*

A operação *retiraProcessoGrupo()* é executada sobre todos os objetos *ServicoReconfiguracao* quando é detectada a falha de um processo monitorado.

A operação *solicitaRetiradaDoParticionamento()*

Operação solicitada ao coordenador por um objeto *ServicoReconfiguracao* que tenha ficado sem acesso ao servidor de nomes durante um período. Caracteriza a volta de um módulo SDF ao grupo. Esta operação ocorre assim que a comunicação com o servidor de nomes é restabelecida.

A operação *recebeSolicitaçãoInsercaoNoGrupo()*

Operação executada sobre o coordenador para solicitar a inclusão de um novo módulo SDF no grupo.

A operação *atualizaHistoricoMaquina()* ()

Esta operação é executada sobre todos os objetos `ServicoReconfiguracao` quando há algum evento significativo nas máquinas que contém módulos SDF.

A operação *atualizaNumeroReconfiguracao()* ()

A operação `atualizaNumeroReconfiguracao()` é executada pelo coordenador sobre todos os objetos `ServicoReconfiguracao` para não permitir entrada de novos membros no grupo até que uma reconfiguração em andamento seja finalizada.

As operações *removeIpTabela()* e *removeIPParticionamento()* ()

Operações que são executadas pelo coordenador, sobre todos os objetos desta classe, quando há o retorno de um módulo SDF ao grupo (executado quando a operação `solicitaRetiradaDoParticionamento()` é disparada sobre o coordenador). Neste caso, os módulos SDF são avisados para retirarem o endereço IP das suas tabelas de máquinas indisponíveis e particionadas. A operação `removeIPTabela()` também é executada quando um novo modulo SDF é colocado em funcionamento no lugar de um módulo que tenha falhado.

A operacao *recebeEstadoModuloCoordenador()*

O coordenador aciona esta operação nos objetos `ServicoReconfiguracao` do grupo, enviando a lista de processos corretos e falhos. O objetivo é que cada módulo confira o seu estado no grupo. Se o módulo estiver em estado igual ou mais adiantado ao coordenador estão ele ignora as atualizações. Se estiver em um estado atrasado ele atualiza suas informações com as do coordenador.

Esta operação foi definida, mas não implementada na versão atual do SDF. Na implementação atual do SDF, se um dos módulos em estado atrasado for o coordenador, ou o candidato a coordenador, a situação será corrigida com a continuação do protocolo. Isso se dá pois este módulo executará o protocolo de acordo para retirada dos processos da máquina falha que ele ainda considera como estando suspeita. Os módulos em estado adiantado, ao receber a solicitação sobre o estado de uma máquina que já teve acordo,

simplesmente concordam que ela seja retirada e desconsideram a mensagem para retirar os processos desta máquina da sua lista de processos corretos.

Como visto, as operações listadas anteriormente são algumas das operações realizadas entre objetos do mesmo tipo (ou seja entre objetos que são instâncias da classe *ServicoReconfiguracao*). Elas visam o diagnóstico dos processos da aplicação.

Existem outras operações que acontecem entre os objetos da classe *ServicoReconfiguracao* e os objetos das classes *ServicoDFGerente*, *ServicoDFAgenteConexao* e *ServicoDFAgenteObjetos*. Estas operações visam à detecção de falhas de processos e máquinas. A seguir, listamos na figura 4.4, algumas destas operações. Elas são necessárias para que as informações de detecção possam chegar até o objeto *ServicoReconfiguracao* e este possa realizar as reconfigurações necessárias e, conseqüentemente, o diagnóstico dos processos. Em seguida apresentaremos a descrição das operações listadas na figura 4.4.

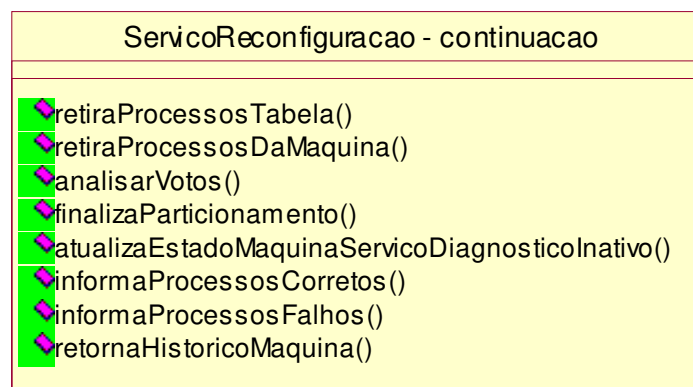


Figura 4.4 - Métodos que são executados pelos objetos do módulo SDF local

A operação *retiraProcessosTabela()*

Esta operação avisa ao objeto *ServicoReconfiguracao* da máquina que um processo local está falho. Ela é disparada pelo objeto da classe *ServicoDFGerente* do módulo.

A operação *retiraProcessosDaMaquina()* ()

Operação que atualiza o estado de uma máquina como suspeita no objeto *ServicoReconfiguracao*. Ela é disparada pelo objeto da classe *ServicoDFGerente* existente no módulo local. Se o objeto *ServicoReconfiguracao* for o coordenador do grupo, então será iniciada uma rodada de votos para confirmar a suspeita de falha da máquina.

A operação *contaModulos()* ()

Esta operação é responsável por descobrir quantos módulos SDF na rede estão ativos no grupo. É executada pelo coordenador, antes de analisar os votos recebidos sobre uma máquina suspeita.

A operação *analisarVotos()* ()

Operação que analisa os votos já recebidos sobre uma determinada máquina para decidir se esta terá seus processos retirados do grupo. Uma *thread* chamada *TemporizadorCoordenador* é que dispara esta operação sobre o coordenador do grupo.

A operação *analisarVotosProximoCoord()* ()

Operação responsável por analisar se houve unanimidade nos votos para eleição de um novo coordenador. Ela é disparada pela *thread* *TemporizadorEleicaoCoordenador* sobre o candidato a novo coordenador.

A operação *finalizaParticionamento()* ()

Esta operação avisa, ao objeto *ServicoReconfiguracao* de uma máquina, que o acesso ao servidor de nomes foi restabelecido. A partir deste momento é possível solicitar uma nova inserção do módulo SDF no grupo. Esta operação é disparada pela *thread* *ThreadMonitoriaServidorNomes*. Esta *thread* fica monitorando o acesso ao servidor de nomes quando a comunicação com este está interrompida temporariamente.

A operação *atualizaEstadoMaquinaServicoDiagnosticoInativo()* ()

Esta operação avisa ao objeto *ServicoReconfiguracao* que um módulo SDF do grupo falhou e que os processos desta máquina estão em estado suspeito. É o objeto *ServicoGerente* que dispara esta operação quando detecta falha no módulo SDF de uma máquina com processo falho.

As operações *informaProcessosCorretos()*, *InformaProcessosFalhos()* e *retornaHistoricoMaquina()*

Operações executadas sobre o objeto *ServicoReconfiguracao* coordenador por uma máquina que esteve sem acesso ao servidor de nomes mas que está apta para voltar ao grupo. O objetivo é obter o estado atual do grupo

4.1.2. Classe *ServicoDFGerente*

A classe *ServicoDFGerente* é responsável por receber as informações de detecção e se certificar do tipo de falha. Sendo detectado um problema em um processo monitorado ou em uma máquina, cabe aos objetos desta classe solicitar investigações adicionais para confirmar a natureza do problema. Confirmando a falha, o objeto *ServicoDFGerente* aciona o objeto *ServicoReconfiguracao*, do seu módulo local, para que este inicie uma reconfiguração. Outra função desta classe é armazenar tabelas contendo todo o histórico das máquinas que fazem parte do grupo e tabelas que contêm os tempos de comunicação atuais entre o módulo SDF e os processos monitorados. Além destas tabelas, que tem valores variáveis conforme a monitoria dos processos, existem outras tabelas com valores fixos definidos. Nelas encontram-se os diferentes níveis de tempo de comunicação, definidos pelos usuários, para cada processo monitorado. São utilizadas para saber se um processo está respondendo dentro dos valores definidos para ele quando este foi inicializado.

A figura 4.5 mostra alguns dos métodos desta classe. Os três primeiros métodos são acionados sobre a plataforma CORBA e os demais são executados por objetos do módulo SDF local.

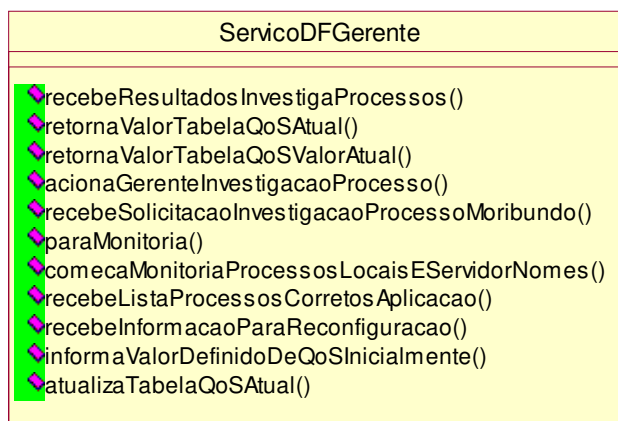


Figura 4.5 - Classe ServicoDFGerente com seus métodos principais

A operação *recebeResultadoInvestigaProcesso()*

Esta operação é disparada pelo objeto ServicoDFAgenteObjetos que foi acionado para realizar uma investigação. Este objeto informa, através desta operação, o estado do processo suspeito.

A operação *retornaValorTabelaQoSValorAtual()* e a operação *retornaValorTabelaQoSAtual()*

São operações que retornam valores existentes nas tabelas do objeto ServicoDFGerente sobre o tempo de comunicação entre o módulo SDF local e os processos monitorados. Estas operações são executadas pelos objetos do Visualizador para informar aos usuários a situação dos processos da aplicação.

A operação *acionaGerenteInvestigacaoProcesso()*

Operação executada pelo objeto ServicoDFAgenteConexao quando este detecta um problema em um processo monitorado ou em uma máquina onde existe módulo do SDF. Quando recebe esta chamada o objeto ServicoDFGerente começa a investigação para descobrir a origem do problema.

A operação *recebeSolicitacaoInvestigacaoProcessoMoribundo()*

Operação solicitada pelo objeto ServicoDFAgenteObjetos sobre o objeto ServicoDFGerente do módulo SDF local. Ela é acionada quando um processo está na tabela

do Sistema operacional mas não responde a solicitações remotas. O objeto `ServicoDFGerente`, ao receber esta chamada, faz requisições ao processo suspeito para ver se este está respondendo às chamadas locais.

A operação *paraMonitoria()*

Operação executada pelo objeto `ServicoReconfiguracao` quando há interrupção na comunicação com o servidor de nomes. O objetivo é parar a monitoria dos processos até que o acesso ao servidor de nomes seja reestabelecido. O objeto `ServicoDFGerente` ao receber esta solicitação avisa ao objeto `ServicoDFAgenteConexao` que pare a sua monitoria.

A operação *comecaMonitoriaProcessosLocaisEServidorNomes()*

Operação que inicializa uma monitoria à máquina que contém o servidor de nomes. Esta solicitação é feita pelo objeto `ServicoReconfiguracao`.

A operação *recebeListaProcessosCorretosAplicacao()*

Esta operação é executada no momento da inicialização do módulo SDF local e tem por objetivo obter os processos corretos do grupo. Esta operação ocorre também após o restabelecimento do contato com o servidor de nomes.

O objeto `ServicoReconfiguracao` é quem executa esta operação passando como parâmetro a nova lista de processos a serem monitorados.

A operação *recebeInformacaoParaReconfiguracao()*

A operação `recebeInformacaoParaReconfiguracao()` é executada pelo objeto `ServicoDFAgenteObjetos` sobre objetos `ServicoDFGerente`, quando é detectado que um processo já não está na tabela do Sistema operacional. Esta operação também é executada pelo próprio objeto `ServicoDFGerente` quando este detecta que um processo não responde a requisições locais, apesar de estar no sistema operacional.

A operação *informaValorDefinidoDeQoSInicialmente()*

A operação `informaValorDefinidoDeQoSInicialmente()` é executada pelo objeto `ServicoDFAgenteConexao` sobre um objeto da classe `ServicoDFGerente` para saber os

valores de tempo de comunicação definidos inicialmente para o processo que esta sendo investigado.

A operação *atualizaTabelaQoSAtual* ()

A operação *atualizaTabelaQoSAtual* () é executada por um objeto *ServicoDFAgenteConexao* sobre o objeto *ServicoDFGerente* do módulo local para atualizar o valor atual de tempo de comunicação com um processo.

4.1.3. Classe *ServicoDFAgenteConexao*

Esta classe é responsável pela monitoria constante dos processos a fim de detectar qualquer falha que aconteça. Seus objetos fazem isso por comparar valores definidos de tempo de comunicação(TC) entre o módulo local e os processos com os valores calculados encontrados na monitoria do processo. Para cada processo monitorado foram definidos três níveis de tempos de comunicação (TC1, TC2 e TC3) que representam três faixas de qualidade de serviço. Como visto na arquitetura apresentada no capítulo 3, se, durante a monitoria do SDF, o valor encontrado de tempo de comunicação for superior aos valores existentes em TC1, TC2 e TC3, ou se o processo não responder, ele é marcado como suspeito.

Para executar suas funções, os objetos da classe *ServicoDFAgenteConexao* têm acesso às tabelas existentes no objeto *ServicoDFGerente* do módulo SDF local. Como as ações de investigação sobre os processos monitorados são todas locais, não existem métodos na classe *ServicoDFAgenteConexao* solicitados via plataforma CORBA.

A figura 4.6 mostra os principais métodos da classe *ServicoDFAgenteConexao* que serão descritos em seguida.

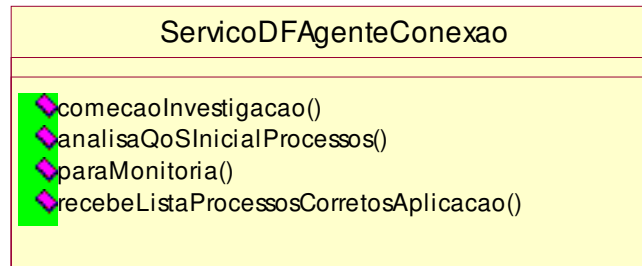


Figura 4.6 - Classe ServicoDFAgenteConexao e seus métodos

A operação *comecaInvestigacao()*

Esta operação é executada para inicializar a monitoria dos processos do grupo. Ela é disparada no momento da inicialização do módulo SDF na máquina.

A operação *analisaQoSInicialProcessos()*

Operação solicitada periodicamente pela *thread* AnalisadorDeConexoes. Quando ocorre esta solicitação, o objeto ServicoDFAgenteConexao analisa o tempo de comunicação encontrado (TC) entre o módulo SDF local e os processos comparando com os tempos de comunicação (TC1, TC2 e TC3) definidos inicialmente para cada processo pelo usuário. Se estes valores estiverem fora dos limites estabelecidos então o processo é considerado suspeito. Se o processo não responder à investigação, o objeto ServicoDFAgenteConexao avisará ao objeto ServicoDFGerente do módulo local.

A operação *paraMonitoria()*

Operação solicitada quando há interrupção no acesso ao servidor de nomes. A monitoria dos processos deve ser temporariamente finalizada até que a situação seja resolvida. Esta operação é acionada pelo objeto ServicoDFGerente do módulo SDF local após ele mesmo ter recebido uma solicitação para parar a monitoria.

A operação *recebeListaProcessosCorretosAplicacao()*

No momento da inicialização do módulo SDF esta operação é executada para associar o objeto desta classe com o objeto ListaProcessosCorretosAplicacao. Sobre os processos desta lista é que o objeto ServicoDFAgenteConexao fará as investigações. Esta operação acontece também após o restabelecimento do acesso ao servidor de nomes.

4.1.4. Classe ServicoDFAgenteObjetos

A classe ServicoDFAgenteObjetos é responsável por investigar no sistema operacional o estado de um processo que não está respondendo às requisições remotas.

Na figura 4.7 vemos a classe ServicoDFAgenteObjetos com o seu método principal que é executado sobre a plataforma CORBA. A seguir, a descrição do método investigaSOProcesso().

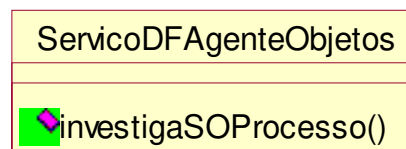


Figura 4.7 - Classe ServicoDFAgenteObjetos com seu principal método

A operação *investigaSOProcesso()*

Esta operação é solicitada por objetos da classe ServicoDFGerente via plataforma CORBA. A operação dispara uma investigação no sistema operacional que retorna se o processo suspeito está ativo ou se está falho. A investigação é feita através de um método nativo Java que acessa uma DLL (*Dynamic Link Library*) em C, InvestigaTabelaSO.c. Esta DLL faz investigações no sistema operacional através da API (*Application Programming Interface*), “psapi.h”, do Windows NT.

A figura 4.8 mostra um trecho do código da DLL InvestigaTabelaSO.c que a operação investigaSOProcesso() chama para verificar se o processo está ativo no sistema operacional. Ao fazer a solicitação, o PID do processo, que é o seu identificador no sistema operacional, é passado como parâmetro. Este identificador é comparado com os identificadores dos processos ativos na tabela do sistema operacional, conforme pode ser visto na figura 4.8. Se o identificador não corresponder a um identificador de processos ativos, então o retorno a essa chamada será falso. Caso contrário, o retorno será verdadeiro.

```

#include <windows.h>
#include "psapi.h"
#include "ServicoDFAgenteObjetos.h"

BOOL APIENTRY DllMain(HANDLE hmodule, DWORD dwReason, void **lpReserved)
{
return TRUE;
}
JNIEXPORT jboolean JNICALL Java_ServicoDFAgenteObjetos_InvestigaSO
(JNIEnv *jEnv, jobject mythis, jlong pid)
{
    :
    :
    :
    if(!EnumProcesses( aProcesses, sizeof(aProcesses), &cbNeeded))
        return FALSE;
    else {
        cProcesses = cbNeeded / sizeof(DWORD);
        for (i=0; i<cProcesses; i++){
            if (pid==aProcesses[i]){
                resultado=TRUE;
            }
        }
    }
    return resultado;
}
}

```

Retorna os processos ativos no Sistema Operacional

Descobre quantos processos estão ativos no Sistema Operacional

Testa para ver se o processo investigado é um dos processos ativos no Sistema Operacional

Figura 4.8 – Trecho do código da DLL InvestigaTabelaSO.c

Quando o processo não responde a solicitações remotas, mas está no sistema operacional, o objeto ServicoDFGerente é acionado para executar chamadas locais ao processo. Estas chamadas locais visam determinar se o processo ainda está ativo.

Para que o objeto ServicoDFAgenteObjetos possa fazer consultas sobre o estado de processos em outros sistemas operacionais é necessário o desenvolvimento de métodos nativos Java que utilizem APIs específicas dos sistemas operacionais em questão.

4.1.5. Classe ListaProcessosCorretosAplicacao

A classe ListaProcessosCorretosAplicacao é responsável por guardar os dados referentes aos processos que devem ser monitorados pelo serviço de diagnóstico. Em cada módulo SDF existe um objeto desta classe que é criado pelo objeto ServicoReconfiguracao e compartilhado com os outros objetos do módulo SDF local. Tem como atributos vetores que contém informações de cada um dos processos monitorados.

Na figura 4.9 podemos ver os atributos da classe `ListaProcessosCorretosAplicacao`. O atributo `ListaProcesso` é um vetor que contém as referências aos processos corretos que foram inseridos para serem monitorados. Os atributos, que também são vetores, `ListaProcessoID`, `ListaProcessoPID`, `ListaProcessoIP`, `ListaProcessoTipo`, `ListaProcessoSuspeito` e `EstadoMaquina` referem-se, respectivamente, aos identificadores dos processos no SDF, aos identificadores no sistema operacional (PIDs), aos endereços IP das máquinas onde os processos residem, ao seu tipo (conforme será explicado no item 4.3), a indicadores que determinam se o processo está em estado suspeito e a indicadores que determinam se a máquina está falha. Além dos atributos descritos acima, temos também como atributos o número total de processos corretos que o SDF pode monitorar (`numprocessos`) e o número atual que esta sendo monitorado (`chave`), conforme mostra a figura 4.9. O número total de processos corretos que podem ser monitorados é definido pelo usuário do serviço quando o primeiro módulo de uma aplicação é inicializado.










ListaProcessosCorretosAplicacao	
	<code>ListaProcesso[] : Processo</code>
	<code>ListaProcessoID[] : int</code>
	<code>ListaProcessoPID[] : long</code>
	<code>ListaProcessoIP[] : String</code>
	<code>ListaProcessoTipo[] : String</code>
	<code>ListaProcessoSuspeito[] : int</code>
	<code>EstadoMaquina[] : boolean</code>
	<code>numProcesso : int</code>
	<code>chave : int = 0</code>

Figura 4.9 - Atributos da classe `ListaProcessosCorretosAplicacao`

Algumas das operações definidas para os objetos que são instâncias da classe `ListaProcessosCorretosAplicacao` são mostradas na figura 4.10 e estão descritas a seguir.

ListaProcessosCorretosAplicacao
retornaProcessos()
insereProcesso()
atualizaVetor()
removeProcesso()
retornaTabela()
retornaTabelaID()
retornaTabelaSuspeito()
retornaTabelaPID()
retornaTabelaIP()
retornaTabelaEstadoMaquina()
retornaTabelaTipos()
atualizaSeProcessoSuspeito()

Figura 4.10 - Métodos da classe ListaProcessosCorretosAplicacao

A operação *retornaProcessos()*

Esta operação retorna o vetor ListaProcesso que contém a referência a todos os processos da aplicação que estão sendo monitorados.

A operação *insereProcesso()*

Esta operação insere um novo processo no grupo. Envolve receber a referência ao objeto, o seu valor de identificação no SDF, o valor do PID no sistema operacional, o endereço IP da máquina e o tipo, armazenando-os, respectivamente, nos vetores ListaProcesso, ListaProcessoID, ListaProcessoPID, ListaProcessoIP, e ListaProcessoTipo. São os objetos da classe ServicoReconfiguracao que manipulam os objetos desta classe e executam o método de inserção de um novo processo na lista.

A operação *atualizaVetor()*

Quando um módulo SDF é introduzido em um grupo já em andamento, ou então quando uma máquina retorna ao grupo, é necessário obter a situação dos processos monitorados. É o objeto ServicoReconfiguracao que faz esta solicitação, recebendo em seguida as informações dos processos do grupo.

A operação *removeProcesso()*

A operação *removeProcesso()* retira as referências de um processo falho do grupo. Esta operação é executada pelo objeto *ServicoReconfiguracao* sobre o objeto *ListaProcessosCorretosAplicacao* quando há falha de um processo.

As operações *retornaTabela()*, *retornaTabelaID()*, *retornaTabelaSuspeito()*, *retornaTabelaPID()*, *retornaTabelaIP()*, *retornaEstadoMaquina()*, *retornaTabelaTipo()*

Operações que retornam, respectivamente, os vetores *ListaProcesso*, *ListaProcessoID*, *ListaProcessoSuspeito*, *ListaProcessoPID*, *ListaProcessoIP*, *EstadoMaquina*, *ListaProcessoTipo*, de um objeto *ListaProcessosCorretosAplicacao*.

A operação *atualizaSeProcessoSuspeito()*

Esta operação atualiza um processo como suspeito. É o objeto da classe *ServicoDFAgenteConexao* que solicita esta atualização.

4.1.6. Classe *ListaProcessosFalhosAplicacao*

Esta classe é responsável por guardar os dados referentes aos processos que falharam durante a monitoria do serviço de diagnóstico. Em cada módulo SDF existe um objeto do tipo *ListaProcessoFalhosAplicacao* que é criado pelo objeto *ServicoReconfiguracao*. Os objetos desta classe têm os mesmos atributos da classe *ListaProcessosCorretosAplicacao*. Existe, no entanto, um atributo adicional que é o vetor *ListaProcessosfalha* que contém o motivo da falha dos processos existentes neste objeto.

A classe *ListaProcessosFalhosAplicacao* possui métodos de manipulação de processos iguais aos existentes na classe *ListaProcessosCorretosAplicacao*.

Na figura 4.11 vemos os atributos desta classe e alguns métodos da classe *ListaProcessosFalhosAplicacao*. Descreveremos, na seqüência, apenas os métodos que não existem na classe *ListaProcessosCorretosAplicacao*.

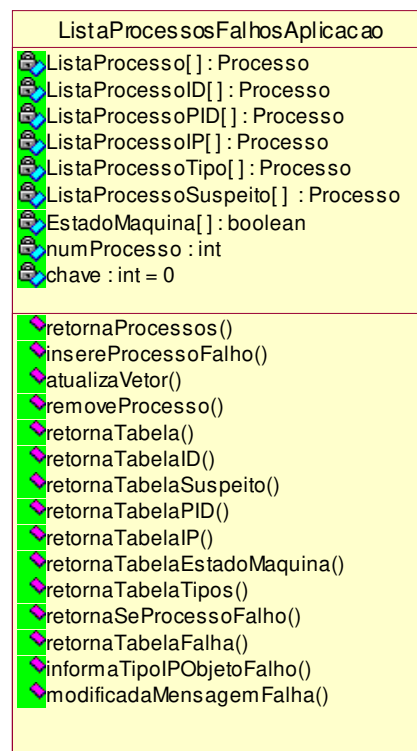


Figura 4.11 - Classe ListaProcessosFalhosAplicacao com seus principais métodos e os atributos

A operação *retornaSeProcessoFalho()*

Esta operação retorna se um processo, cujo identificador no sistema operacional (PID) é passado como parâmetro, está no objeto ListaProcessosFalhosAplicacao.

A operação *retornaTabelaFalha()*

Operação que retorna o vetor ListaProcessosFalha.

A operação *retornaTipoIPObjetoFalho()*

Esta operação retorna a informação do tipo do objeto que falhou e em que endereço IP este residia. Esta informação será utilizada pelo Visualizador.

A operação *modificadaMensagemFalha()*

A operação modificadaMensagemFalha(), como o próprio nome diz, modifica a descrição da mensagem de falha de um processo. Esta mudança é necessária quando uma máquina

que estava sem acesso ao servidor de nomes retorna ao grupo mas tem processos que falharam durante o período de particionamento.

4.2. Outras Classes que Auxiliam as Classes do Núcleo Básico

Além das classes do núcleo básico, foram implementadas classes adicionais para que o SDF realize as atividades previstas na sua arquitetura. Neste item descreveremos resumidamente estas classes, salientando como elas interagem com as classes do núcleo básico do serviço.

4.2.1. Classe AnalisadorDeConexoes

Quando o módulo SDF é inicializado em uma máquina, um objeto desta classe é instanciado também. Este objeto dispara, periodicamente, uma requisição ao objeto ServicoDFAgenteConexao para que este investigue o estado dos processos que estão no objeto ListaProcessosCorretosAplicacao. A seguir, vemos a figura 4.12 que mostra a classe AnalisadorDeConexoes.

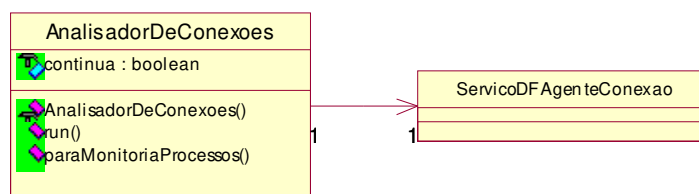


Figura 4.12 - Classe AnalisadorDeConexao

4.2.2. Classe Pinger

Instâncias da classe Pinger são criadas pelo objeto ServicoDFAgenteConexao para testar o estado de uma máquina. Estas solicitações são feitas através da operação `doPing()`, que

envia uma mensagem de teste para a porta ECHO da máquina a ser testada e espera uma resposta. Além do método doPing(), existem outros métodos especificados nesta classe conforme pode ser visto na figura 4.13.

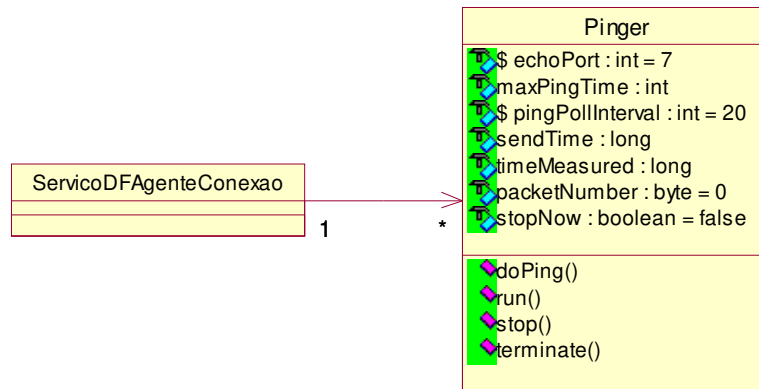


Figura 4.13 - Classe Pinger

4.2.3. Classe ListaExcluidosParticionamento

Um objeto desta classe é instanciado quando o módulo SDF é inicializado. No entanto, este objeto só é utilizado quando há interrupção na comunicação com o servidor de nomes.

Nesta classe existem atributos e métodos semelhantes aos que encontramos nas classes ListaProcessosCorretosAplicacao e ListaProcessosFalhosAplicacao. Seu objetivo é guardar os processos locais que estão corretos mas que foram considerados falhos pelo grupo durante a interrupção da comunicação com o servidor de nomes. Quando a comunicação é restabelecida, estes processos são inseridos novamente nos objetos ListaProcessosCorretosAplicacao para voltarem a ser monitorados.

Na figura 4.14 podemos ver os principais atributos e métodos desta classe.

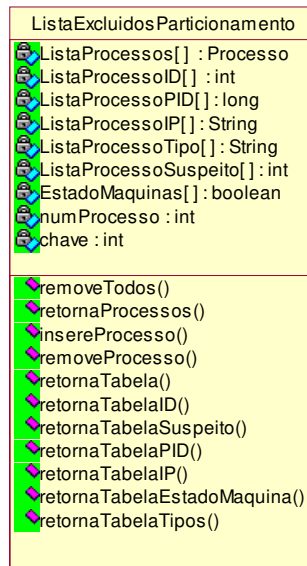


Figura 4.14 - Classe ListaExcluidosParticionamento

4.2.4. Classe ThreadSolicitacaoInvestigacaoProcessoSO

Instâncias desta classe são criadas pelo objeto ServicoDFGerente quando é necessário solicitar uma investigação a um objeto ServicoDFAgenteObjetos sobre um processo em estado suspeito.

Na figura 4.15 vemos a classe ThreadSolicitacaoInvestigacaoProcessoSO e seu relacionamento com a classe ServicoDFGerente.

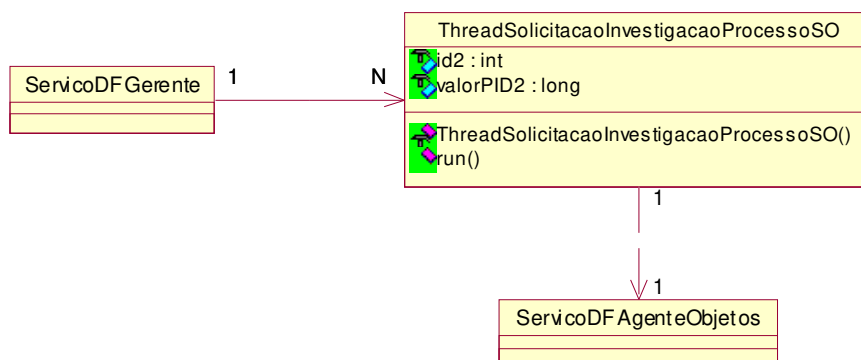


Figura 4.15 - Classe ThreadSolicitaInvestigacaoProcessos e classe ServicoDFGerente

4.2.5. Classe ThreadMonitoriaServidorNomes

Um objeto da classe `ThreadMonitoriaServidorNomes` é criado pelo objeto `ServicoReconfiguracao` quando ocorre uma interrupção na comunicação com o servidor de nomes. Esta classe é subclasse de `java.lang.Thread` e sua função é perceber quando a conexão com o servidor de nomes é restabelecida.

Quando há o restabelecimento da comunicação entre a máquina e o servidor de nomes, o objeto `ServicoReconfiguracao` do módulo local é avisado. Ele executa as ações necessárias para que os processos locais sejam inseridos novamente nos objetos `ListaProcessosCorretosAplicacao` do grupo. Em seguida a *thread* é finalizada.

A figura 4.16 apresenta a classe `ThreadMonitoriaServidorNomes` e seu relacionamento com a classe `ServicoReconfiguracao`.

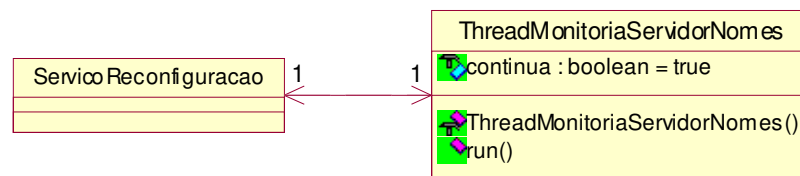


Figura 4.16 - Classe `ThreadMonitoriaServidorNomes`

4.2.6. Classe ThreadAtualizaHistorico e Classe ThreadHistorico

Um objeto da classe `ThreadAtualizaHistorico` é criado pelo coordenador quando há um evento significativo sobre alguma das máquinas do grupo. Se vários eventos de falhas ocorrerem ao mesmo tempo teremos vários objetos da classe `ThreadAtualizaHistorico` trabalhando de forma concorrente.

A classe `ThreadAtualizaHistorico` tem como função disparar *threads* de atualizações, que são instâncias da classe `ThreadHistorico`, para todos os objetos `ServicoReconfiguracao` inscritos no servidor de nomes. Os objetos da classe `ThreadHistorico` têm por função solicitar a atualização do histórico das máquinas do grupo em cada módulo SDF. Sobre a

plataforma CORBA, estes objetos solicitam a execução do método `atualizaHistoricoMaquina()`.

A figura 4.17 mostra as classes `ThreadAtualizaHistorico`, `ThreadHistorico` e o relacionamento destas com os objetos da classe `ServicoReconfiguracao`.

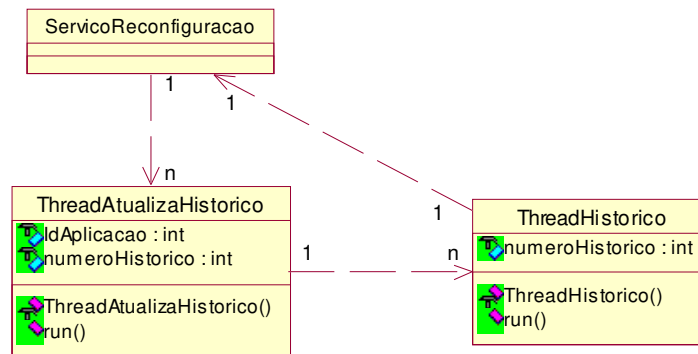


Figura 4.17 -Classes `ThreadAtualizaHistorico`, Classe `ThreadHistorico` e Classe `ServicoReconfiguracao`

4.2.7. Classe `ThreadAtualizaNumeroReconfiguracao` e Classe `ThreadNumeroEstadoReconfiguracao`

A classe `ThreadAtualizaNumeroReconfiguracao` tem como função disparar *threads* de atualizações, que são instâncias de `ThreadNumeroEstadoReconfiguracao`, para todos os módulos SDF inscritos no servidor de nomes.

Um objeto da classe `ThreadAtualizaNumeroReconfiguracao` é criado pelo coordenador do grupo quando uma reconfiguração está sendo inicializada ou quando uma reconfiguração é finalizada. Podem existir reconfigurações de máquinas distintas ocorrendo ao mesmo tempo.

Os objetos da classe `ThreadNumeroEstadoReconfiguracao` acionam, via plataforma CORBA, os objetos `ServidorReconfiguracao` solicitando a estes que executem seu método `atualizaNumeroEstadoReconfiguracao()`.

A figura 4.18 apresenta um diagrama de classes contendo as classes `ThreadAtualizaNumeroReconfiguracao`, `ThreadNumeroEstadoReconfiguracao` e `ServicoReconfiguracao`.

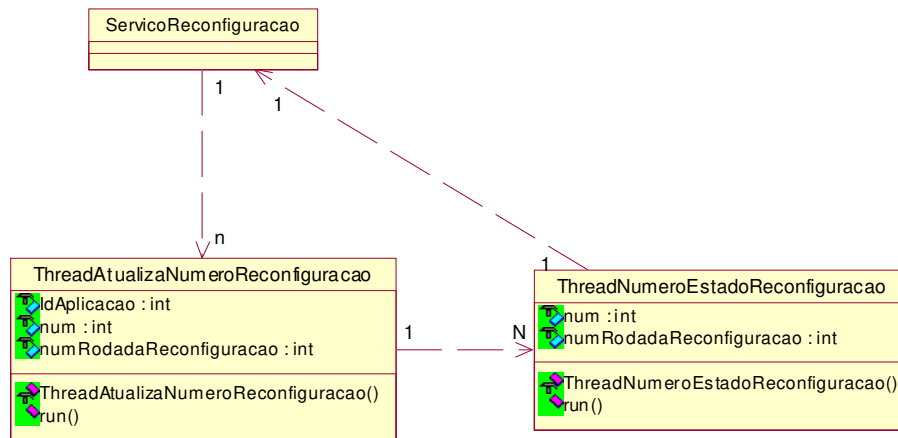


Figura 4.18 - Classe ThreadAtualizaNumeroReconfiguracao, Classe ThreadNumeroEstadoReconfiguracao e Classe ServicoReconfiguracao

4.2.8. Classe ThreadSolicitaVotos e Classe ThreadVotos

Um objeto da classe ThreadSolicitaVotos é instanciado pelo objeto ServicoReconfiguracao coordenador quando é necessário um acordo sobre o estado de uma máquina.

Podem existir vários processos de acordos acontecendo simultaneamente. O objeto ThreadSolicitaVotos percorre o servidor de nomes disparando para cada objeto ServicoReconfiguracao uma *thread*, instância da classe ThreadVotos. Esta *thread* solicitará ao objeto ServicoReconfiguracao que envie seu voto sobre o estado da máquina suspeita. Isto ocorre através da chamada à operação `enviarVotoParaCoordenador()`, sobre plataforma CORBA. A figura 4.19 mostra as classes ServicoReconfiguracao, ThreadSoliciraVotos e ThreadVotos em um diagrama de classes.

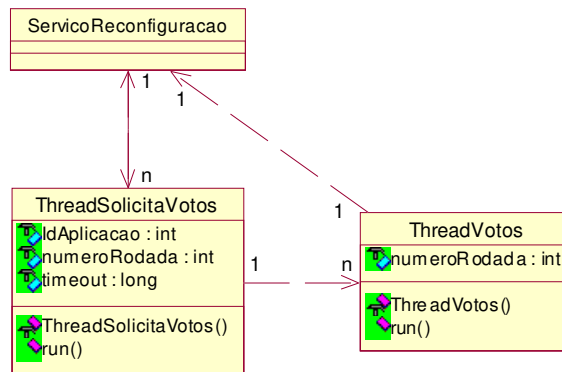


Figura 4.19 - Classe ThreadSolicitaVotos, Classe ThreadVotos e Classe ServicoReconfiguracao

4.2.9. Classe TemporizadorCoordenador

O objeto ServicoReconfiguracao coordenador cria um objeto da classe TemporizadorCoordenador quando existe necessidade de um acordo sobre o estado de uma máquina suspeita. Os objetos desta classe são *threads* que entram em um *timeout* e que depois acionam o coordenador solicitando que este execute sua operação `analisarVotos()`. Na figura 4.20 vemos a classe TemporizadorCoordenador e o relacionamento desta classe com a classe ServicoReconfiguracao.

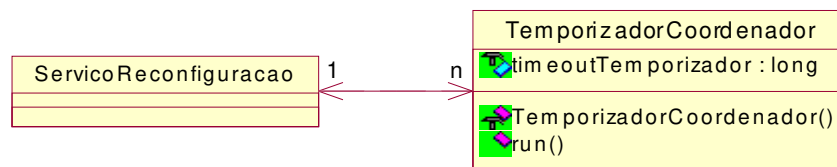


Figura 4.20- Classe Temporizador e Classe ServicoReconfiguracao

4.2.10. Classe ThreadRetiraProcessoGrupo, Classe ThreadSolicitaRetiradaProcessoGrupo e Classe ThreadRetiraProcessosMaquinaFalha

Quando um processo é detectado como falho, o objeto ServicoReconfiguracao da mesma máquina inicia uma rodada para retirá-lo dos objetos ListaProcessosCorretosAplicacao do

grupo. Para isso, um objeto da classe ThreadSolicitaRetiradaProcessoGrupo é criado pelo objeto ServicoReconfiguracao. Ele tem como atribuição percorrer as referências existentes no servidor de nomes disparando uma *thread* do tipo ThreadRetiraProcessoGrupo para cada objeto ServicoReconfiguracao encontrado. Nesta *thread* a operação `retiraProcessoGrupo()` do objeto ServicoReconfiguracao é acionada.

No caso de falha de uma máquina, o objeto ServicoReconfiguracao coordenador cria um objeto ThreadRetiraProcessosMaquinaFalha que dispara *threads*, do tipo ThreadRetiraProcessoGrupo, para cada processo da máquina que falhou em cada objeto ServicoReconfiguracao do grupo.

A figura 4.21 mostra o diagrama de classes envolvendo as classes ThreadRetiraProcessoGrupo, ThreadSolicitaRetiradaProcessoGrupo, ThreadRetiraProcessosMaquinaFalha e ServicoReconfiguracao.

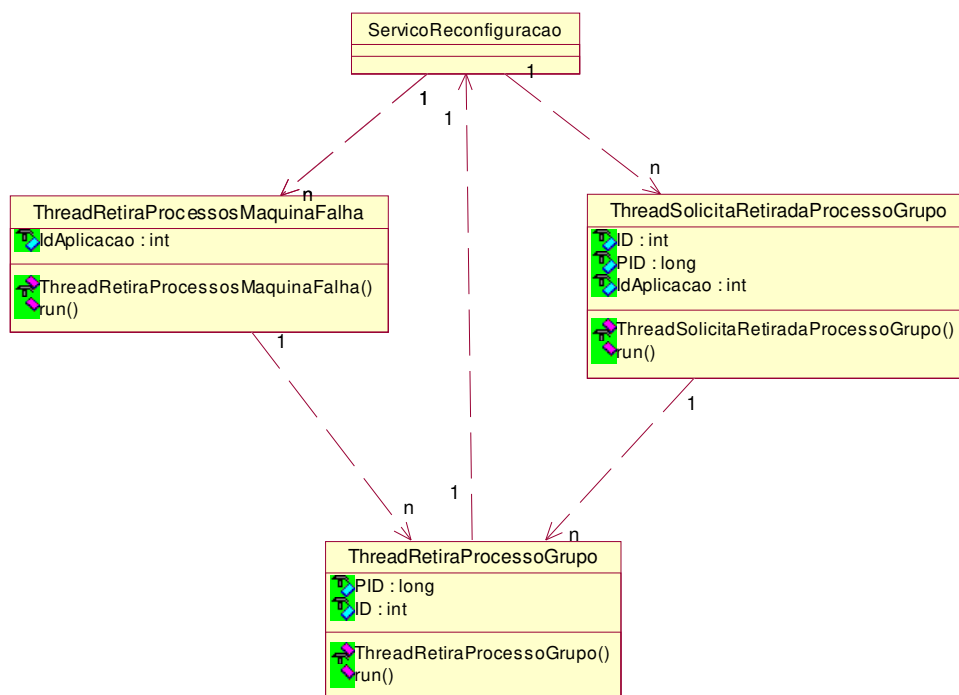


Figura 4.21- ThreadRetiraProcessoGrupo, Classe ThreadSolicitaRetiradaProcessoGrupo, Classe ThreadRetiraProcessosMaquinaFalha e Classe ServicoReconfiguracao

4.2.11. Classe ThreadVotosProximoCoord

Quando há falha do coordenador, o candidato a este cargo solicita a todos os outros objetos `ServicoReconfiguracao` que informem o seu voto para eleição do novo coordenador. Esta solicitação acontece pela criação de instâncias da classe `ThreadVotosProximoCoord`, onde cada uma delas executa a operação `informeVotoProximoCoordenador()`, via plataforma CORBA, sobre os objetos `ServicoReconfiguracao` do grupo. Neste caso, quem faz toda a atividade de obter a referência no servidor de nomes e instanciar uma nova *thread* do tipo `ThreadVotosProximoCoord` é o próprio objeto `ServicoReconfiguracao` candidato a coordenador e não uma *thread* independente como nos exemplos anteriores. Isto se dá pois a prioridade do SDF, no caso de uma falha do coordenador, é estabelecer um novo coordenador, não sendo possível que outras reconfigurações aconteçam até que esta tenha sido finalizada.

Na figura 4.22 podemos ver as classes `ThreadVotosProximoCoord` e `ServicoReconfiguracao`.

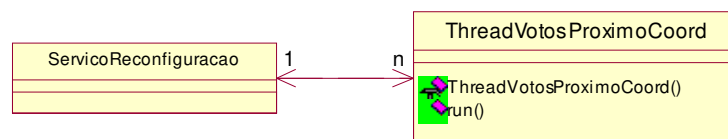


Figura 4.22 - Classe `ThreadVotosProximoCoord` e Classe `ServicoReconfiguracao`

4.2.12. Classe TemporizadorEleicaoCoordenador

O objeto `ServicoReconfiguracao` candidato a coordenador cria um objeto desta classe após solicitar os votos para sua eleição. Este objeto `TemporizadorEleicaoCoordenador` é uma *thread* que aguarda um *timeout* e depois solicita ao candidato a coordenador que analise os votos recebidos. Esta *thread* executa sobre o candidato a coordenador a operação `analizarVotosProximoCoord()`.

A figura 4.23 mostra a classe `TemporizadorEleicaoCoordenador` e seu relacionamento com a classe `ServicoReconfiguracao`.

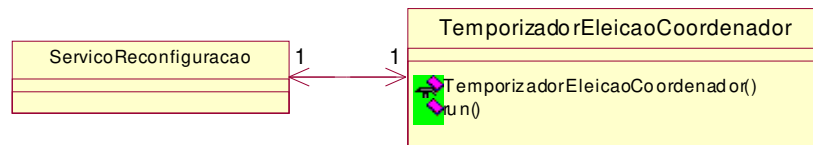


Figura 4.23 - Classe TemporizadorEleicaoCoordenador e Classe ServicoReconfiguracao.

4.2.13. Classe ThreadInsereObjetoParaGerenciamento

Quando uma máquina, que estava sem acesso ao servidor de nomes, retorna ao grupo, cada processo existente no objeto ListaExcluidosParticionamento deve ser inserido no objeto ListaProcessoCorretosAplicacao dos módulos SDF. Assim, o objeto ServicoReconfiguracao solicita a criação de uma *thread* do tipo ThreadInsereObjetoParaGerenciamento para cada processo da máquina.

Na figura 4.24 podemos ver o diagrama de classes envolvendo as classes ThreadInsereObjetoParaGerenciamento e ServicoReconfiguracao.

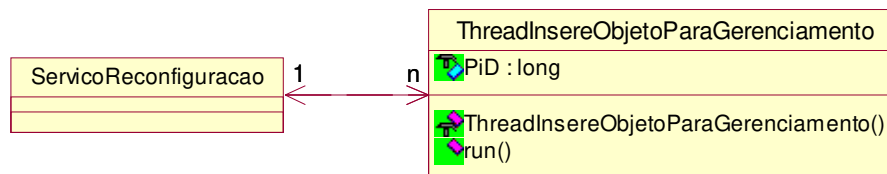


Figura 4.24 - Classe ThreadInsereObjetoParaGerenciamento e Classe ServicoReconfiguracao.

Concluimos aqui a descrição das classes que auxiliam o núcleo básico do SDF. No item 4.3, a seguir, descreveremos a implementação dos tipos de processos que podem ser monitorados pelo serviço de diagnóstico.

4.3. Interfaces de Monitoria e Tipos de Processos Monitorados

Como vimos no item 3.4, uma interface é necessária na arquitetura do SDF para permitir monitoria dos processos da aplicação. Na nossa implementação o SDF monitora processos que implementam uma interface de monitoria chamada *ObjetosSistemas*. Esta interface herda seus métodos de uma interface genérica chamada de *ObjetosGerenciados*. Existe também uma outra interface, *ObjetosRede*, que tem por finalidade permitir a monitoria de objetos que são recursos da rede. Ela também herda da interface *ObjetosGerenciados*. A monitoria de objetos que são recursos da rede pode acontecer através de chamadas ao módulo SNMP existente em tais recursos. Neste trabalho, apesar de propormos uma interface de monitoria para recursos de rede, tais como *hub* e *switch*, estava fora do escopo implementar este tipo de monitoria. Nosso foco foi a monitoria de objetos de sistema. No entanto, a definição de uma interface genérica permitirá futura integração de diferentes tipos de monitoria.

A figura 4.25 mostra as interfaces *ObjetosSistemas*, *ObjetosRedes* e *ObjetosGerenciados*.

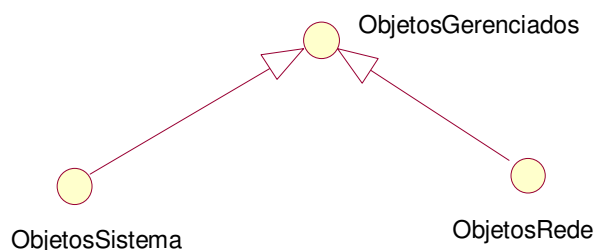


Figura 4.25 - Interfaces de monitoria

A interface *ObjetosSistema* herda três métodos da interface *ObjetosGerenciados*: *insereObjetoParaGerenciamentoSD()*, *insereNovamenteObjetoParaGerenciamentoSD()* e *respondeMonitoria()*. O primeiro método, como o próprio nome diz, permite que o processo em questão seja inserido no objetos *ListaProcessosCorretosAplicacao* de cada um dos módulos SDF do grupo. O segundo método é executado em casos de retorno ao grupo

após ficar sem contato com o servidor de nomes. O terceiro método possibilita que o SDF verifique o estado do processo.

Além dos processos da aplicação que implementam a interface `ObjetosSistema`, o SDF monitora serviços através do uso de processos interfaces, conforme define a arquitetura apresentada no item 3.4.

Um processo interface implementa a interface `ObjetosSistema` e faz a comunicação entre os serviços que não implementam esta interface e o SDF, como pode ser visto na figura 4.26. Nesta figura todos os processos, P1, P2, P3 e P4, implementam a interface `ObjetosSistema`. O processo P1 é um processo interface que auxilia o SDF a monitorar um Serviço Web, P2 representa um processo que pode ser monitorado diretamente pelo SDF, já o processo P3 é um processo interface que monitora um Servidor de Correio, e o processo P4 é um processo interface para um Serviço Java que responde em uma porta qualquer.

O processo interface, que monitora um serviço, deve estar na mesma máquina do serviço monitorado. Esta restrição tem como objetivo evitar tratar das questões de infraestrutura de rede entre o processo interface e o serviço vinculado a este.

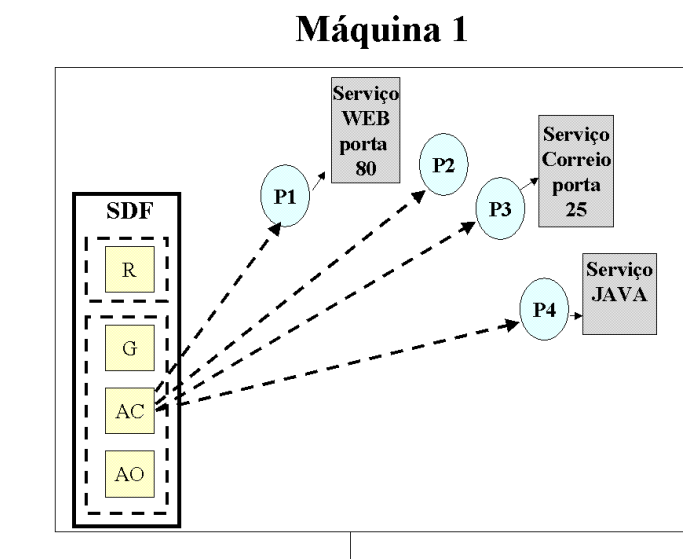


Figura 4.26 - SDF monitorando diversos tipos de processos e serviços.

A figura 4.27, a seguir, mostra as classes que implementam os tipos de processos interface que foram criados nesta versão do SDF. Na figura vemos, também, uma classe

ProcessoCorreto que representa os processos que implementam diretamente a interface ObjetosSistema.

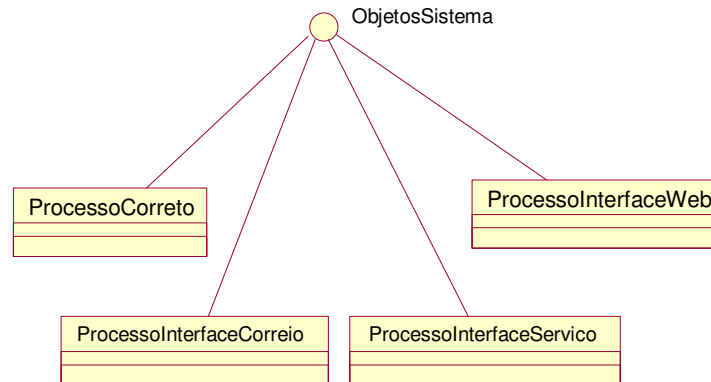


Figura 4.27 - Classes que representam os processos monitorados pelo SDF

No item que se segue, veremos as particularidades das classes apresentadas na figura 4.27.

4.3.1. Classes que Atuam Como Processos Interface

As classes `ProcessoCorreto`, `ProcessoInterfaceWeb`, `ProcessoInterfaceCorreio` e `ProcessoInterfaceServico` respondem às mesmas operações. No entanto, o código existente na operação `respondeMonitoria()` difere entre elas, já que estamos tratando de serviços que reagem de forma diferente à monitoria. Nos sub-itens a seguir veremos as particularidades das classes implementadas para atuar como processos interfaces na monitoria do SDF.

Além destes processos interfaces propostos e implementados, outros podem ser criados para permitir a monitoria de serviços importantes para a aplicação. Poderemos ter processos interfaces para serviços tais como FTP, DNS, NEWS e outros. O princípio por detrás de cada processo interface é o mesmo, no entanto, o código do método `respondeMonitoria()` deve ser implementado de acordo com as características peculiares de cada serviço.

4.3.1.1. Detalhes de Implementação da Classe `ProcessoCorreto`

A classe `ProcessoCorreto` implementa o método `respondeMonitoria()`, definido na interface `ObjetosSistema`, por retornar um valor `true` se o processo estiver ativo.

4.3.1.2. Detalhes de Implementação da Classe `ProcessoInterfaceCorreio`

Os objetos desta classe recebem chamadas ao método `respondeMonitoria()` e estabelecem uma conexão com a porta 25 da máquina do serviço de correio, enviando uma solicitação. Se houver resposta a esta solicitação o serviço é considerado ativo e o método `respondeMonitoria()` do processo interface retorna um valor `true`.

A figura 4.28 mostra o código do método `respondeMonitoria()` da classe `ProcessoInterfaceCorreio`.

```
public synchronized boolean respondeMonitoria(){
    String str;
    Socket t;
    boolean resultado = true;
    InetAddress IA;
    try{
        IA = InetAddress.getLocalHost();
        t = new Socket(IA, 25);
        java.io.DataInputStream is = new
        java.io.DataInputStream(t.getInputStream());
        str = is.readLine();
        t.close();
    }catch(Exception e){
        resultado = false;
    }
    return resultado;
}
```

Figura 4.28 - Método `respondeMonitoria()` da Classe `ProcessoInterfaceCorreio`

4.3.1.3. Detalhes de Implementação da Classe `ProcessoInterfaceWeb`

Um objeto da classe `ProcessoInterfaceWeb` recebe uma mensagem do tipo `respondeMonitoria()`, cria uma conexão com a porta 80 da máquina onde reside o serviço Web e faz requisições HTTP a este serviço. Se houver resposta então o serviço Web é considerado ativo e o método `respondeMonitoria()` do processo interface Web retorna um valor `true`, como mostra o código da figura 4.29.

```
public synchronized boolean respondeMonitoria(){
    String str;
    Socket t;
    boolean resultado = true;
    InetAddress IA;
    try{
        IA = InetAddress.getLocalHost();
        t = new Socket(IA, 80);
        java.io.DataInputStream is = new
        java.io.DataInputStream(t.getInputStream());
        PrintStream out = new
        PrintStream(t.getOutputStream());
        out.println("GET / HTTP/1.0 \n\n");
        str = is.readLine();
        t.close();
    }catch(Exception e){
        resultado = false;
    }
    return resultado;}
}
```

Figura 4.29 - Método `respondeMonitoria()` da Classe `ProcessoInterfaceWeb`

4.3.1.4. Detalhes de Implementação da Classe `ProcessoInterfaceServico`

Os objetos desta classe estabelecem uma conexão com a porta do Serviço Java e enviam uma requisição quando o método `respondeMonitoria()` é acionado. Se houver resposta então o Serviço Java é considerado ativo e o método `respondeMonitoria()`, do processo interface `servico`, retorna um valor `true`.

Na figura 4.30, vemos o código do método `respondeMonitoria()` da classe `ProcessoInterfaceServico`.

```
public synchronized boolean respondeMonitoria(){
    String str;
    Socket t;
    boolean resultado = true;
    InetAddress IA;
    try{
        IA = InetAddress.getLocalHost();
        t = new Socket(IA, portaServico);
        java.io.DataInputStream is = new
java.io.DataInputStream(t.getInputStream());
        str = is.readLine();
        t.close();
    }catch(Exception e){
        resultado = false;
    }
    return resultado;
}
```

Figura 4.30 - Método *respondeMonitoria()* da Classe *ProcessoInterfaceServico*

4.4. Classes que Compõem o Visualizador do Serviço de Diagnóstico de Falhas

Como visto no capítulo 3, o SDF possui uma ferramenta de visualização que possibilita aos usuários terem acesso às informações de diagnóstico dos processos monitorados. Este Visualizador apresenta o estado dos processos da aplicação e o histórico de todas as máquinas do grupo.

As classes que implementam o Visualizador podem ser vistas na figura 4.31 a seguir. Quando este é inicializado, uma instância da classe Visualizador é criada. A depender da visão que deve ser mostrada ao usuário, o Visualizador cria um objeto do tipo VisaoGrupo (para mostrar a situação de todos os processos do grupo), ou um objeto do tipo Historico (para apresentar o histórico das máquinas do grupo). Cada tipo de visão é mostrada em uma tela distinta e possui uma *thread* que fica atualizando as informações periodicamente. A *thread* associada ao objeto VisaoGrupo é a ThreadEstado, conforme pode ser visto na figura 4.31. Ela obtém as informações existentes nas tabelas do objeto ServicoDFGerente e repassa para o objeto VisaoGrupo, onde as informações sobre o estado de todos os processos serão disponibilizada para os usuários. A *thread* ThreadEstadoHist, por sua vez,

pega as informações de histórico das máquinas do grupo, no objeto ServicoReconfiguracao, repassando para o objeto Histórico que disponibilizará estas informações aos usuários.

A figura 4.31 a seguir apresenta o diagrama de classes do Visualizador, mostrando cada uma das classes descritas anteriormente e seus relacionamentos com as classes ServicoDFGerente e ServicoReconfiguracao.

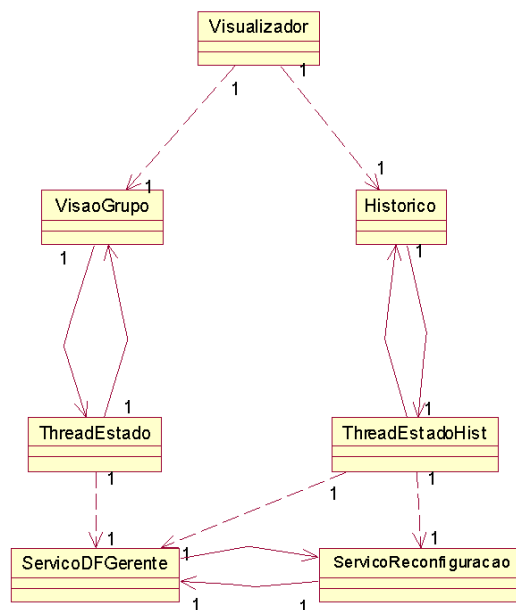


Figura 4.31 - Diagrama de classes do Visualizador e as classes ServicoDFGerente e ServicoReconfiguracao

4.5. Diagramas de Seqüência

Considerando os tipos de falhas descritas no item 3.5 desta dissertação, mostraremos os diagramas de seqüência que representam as atividades executadas pelos objetos do SDF quando detecta estas falhas.

Nos itens 4.5.1 até 4.5.4, a seguir, apresentaremos estes diagramas. No item 4.5.5, mostraremos o diagrama de seqüência que descreve as ações executadas durante o protocolo de acordo apresentado no capítulo 3 (item 3.6). Simplificamos os diagramas mostrando apenas as atividades principais para tornar a visão da seqüência mais clara e de fácil entendimento.

Descrevemos na dissertação apenas os diagramas que consideramos mais importantes para o entendimento do SDF. Outros diagramas, tais como o diagrama de eleição de um novo coordenador, atualização de informação de histórico, não foram apresentados por acrescentarem um nível de detalhe que não contribui para o entendimento do serviço como um todo.

4.5.1. Processo não Responde Dentro dos Limites de Tempo de Comunicação Especificado

O diagrama de seqüência, existente na figura 4.32, apresenta o objeto *ServicoDFAgenteConexao* monitorando um processo, *respondeMonitoria()*, e solicitando a seu objeto *Pinger* que monitore a máquina onde este processo reside, *doPing()*. O valor de tempo de comunicação (TC) encontrado na monitoria do processo é comparado com os valores existentes na tabelas do objeto *ServicoDFGerente*, *informarValorDefinidoQoSInicialmente()*. Se o processo não estiver respondendo dentro dos níveis definidos para ele (TC1, TC2 e TC3) então ele é atualizado como suspeito, *atualizaSeProcessoSuspeito()*. O novo valor encontrado de tempo de comunicação é informado pelo objeto *ServicoDFAgenteConexao* ao objeto *ServicoDFGerente*, através do método *atualizaTabelaQoSAtual()*.

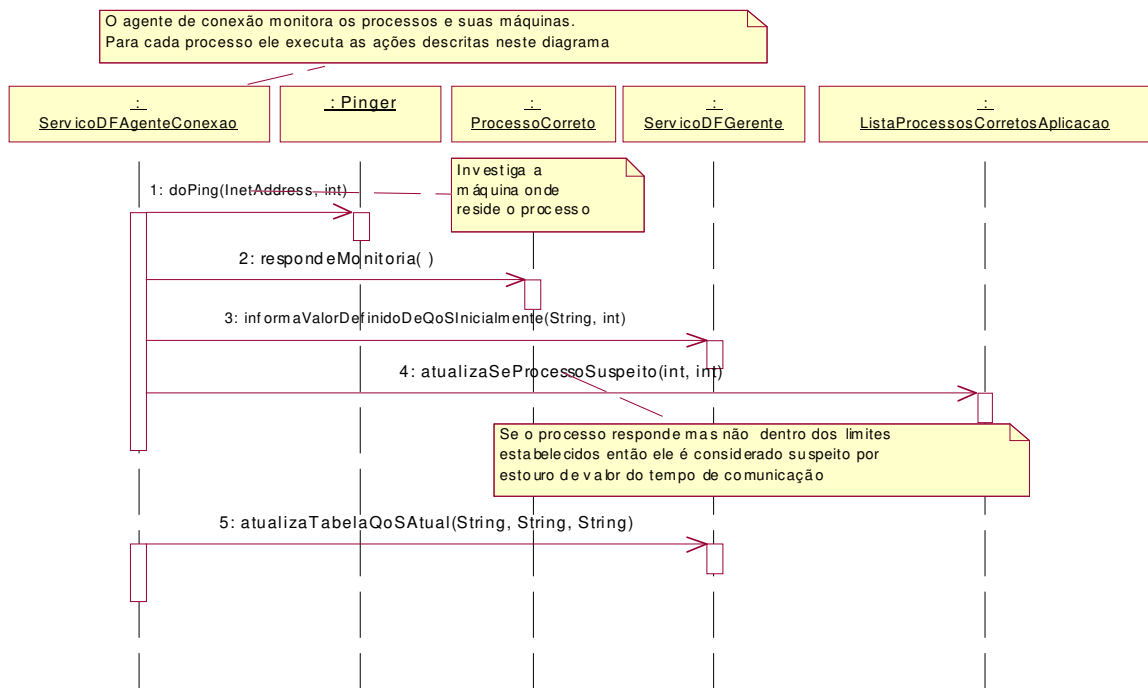


Figura 4.32 - Diagrama : Processo responde solicitações fora dos níveis definidos

4.5.2. Processo não Está Mais na Tabela do Sistema Operacional ou não Responde ao Módulo do SDF Local

As atividades executadas neste caso foram divididas em três diagramas, conforme pode ser visto na figura 4.33, 4.34 e 4.35.

O primeiro diagrama, figura 4.33, compreende a seqüência cronológica desde o momento em que o objeto *ServicoDFAgenteConexao* aciona o objeto *ServicoDFGerente* solicitando mais investigações sobre um processo que não responde, através do método *acionaGerenteInvestigacaoProcesso()*, até o momento em que o objeto *ServicoReconfiguracao* toma conhecimento da falha, *retiraProcessoTabela()*. Os passos desta seqüência incluem a criação de uma *thread* de investigação, *ThreadSolicitacaoInvestigacaoProcessoSO()*, que solicitará ao objeto *ServicoDFAgenteObjetos* investigar o estado do processo suspeito no Sistema Operacional, *investigaSOProcesso()*. O objeto *ServicoDFAgenteObjetos* está na máquina onde o

processo suspeito reside. Se ele detectar que o processo não está no sistema operacional, ou que ele não responde a solicitações locais, *investigaProcessoMoribundo()*, então esta informação é repassada para o objeto *ServicoDFGerente* do módulo SDF local, *recebeInformacaoParaReconfiguracao()*, e este se encarrega de informar a falha ao objeto *ServicoReconfiguracao*, *retiraProcessoTabela()*.

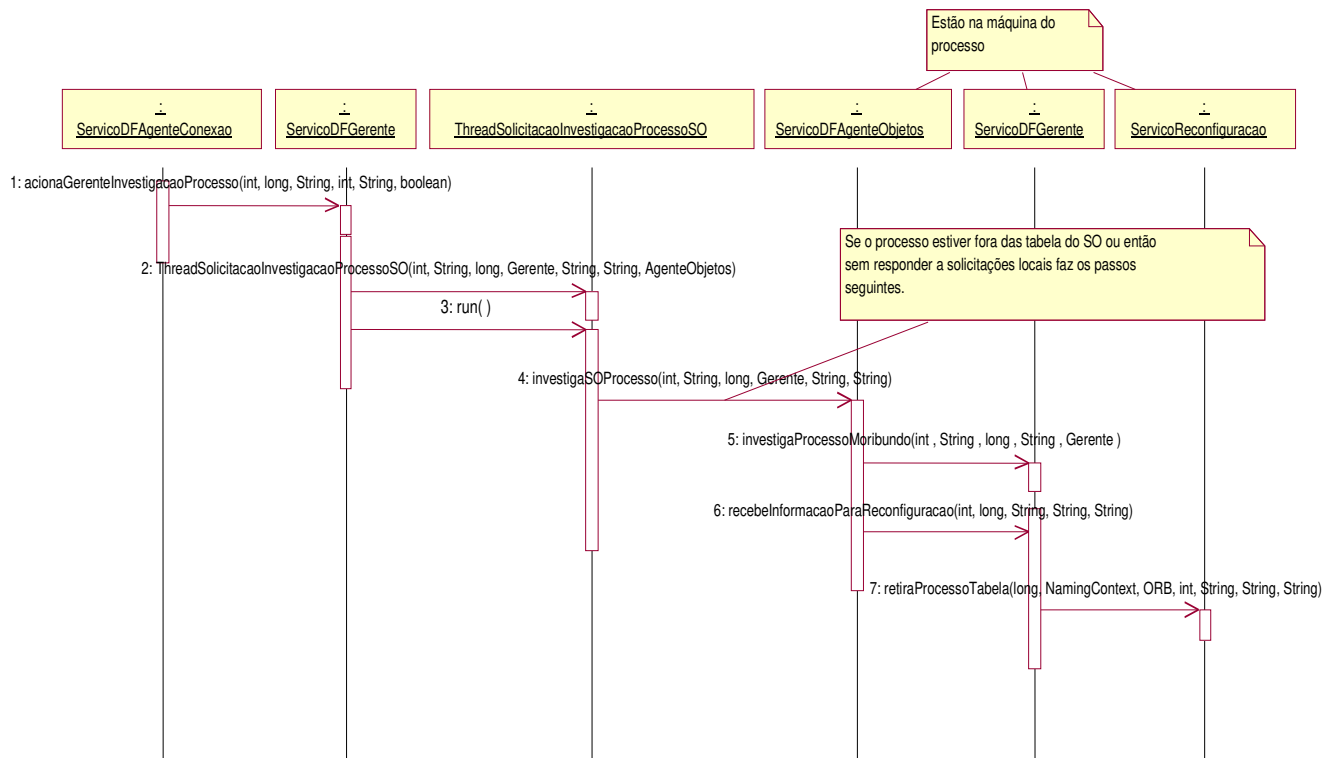


Figura 4.33 - Diagrama : Detecção de processo falho

O segundo diagrama, figura 4.34, continua a seqüência apresentada no diagrama anterior. O objeto *ServicoReconfiguracao* cria uma *thread*, *ThreadSolicitaRetiradaProcessoGrupo()*, que instanciará, para cada módulo SDF, uma outra *thread* *ThreadRetiraProcessosGrupo()*, solicitando que o objeto *ServicoReconfiguracao* do módulo SDF retire o processo falho do grupo, através do método *retiraProcessoGrupo()*.

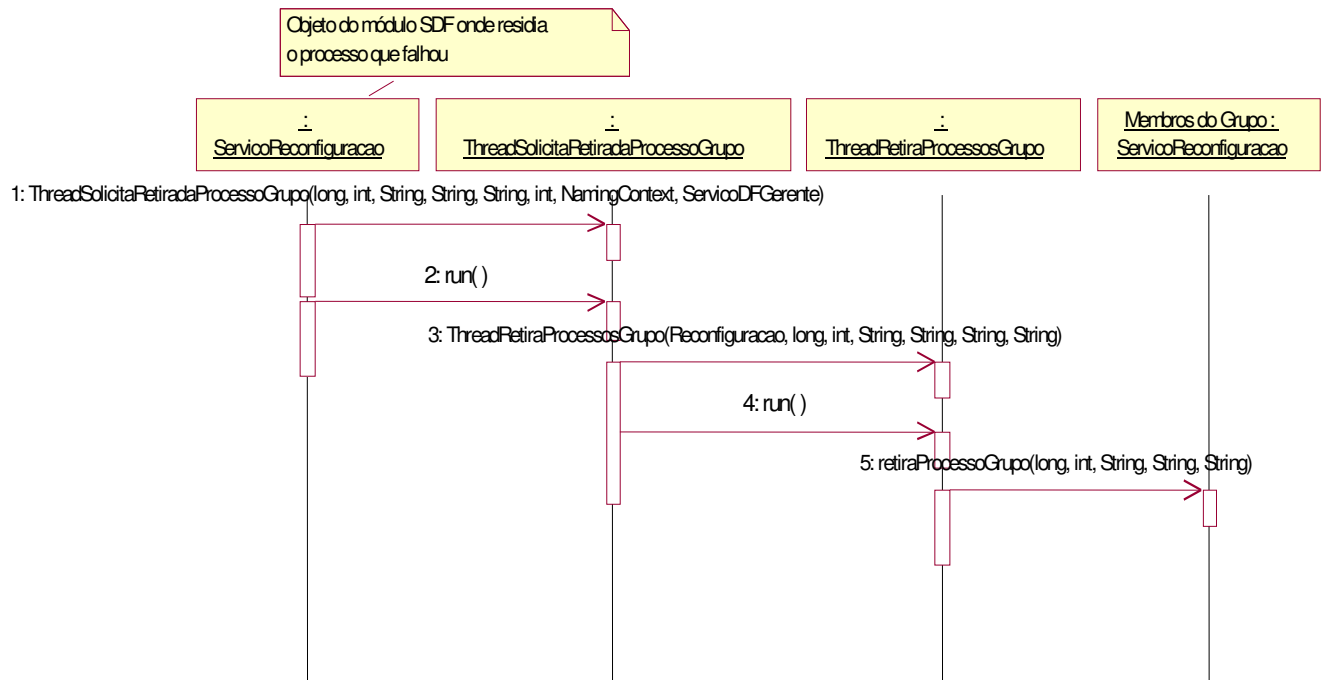


Figura 4.34 - Diagrama : Solicitação de retirada do processo falho

O terceiro diagrama, figura 4.35, compreende as atividades a serem executadas em cada máquina que recebe uma solicitação de retirada de processo. O processo falho deve ser retirado do objeto ListaProcessosCorretosAplicacao, *removeProcesso()*, e colocado no objeto ListaProcessosFalhosAplicacao, *insereProcessoFalho()*.

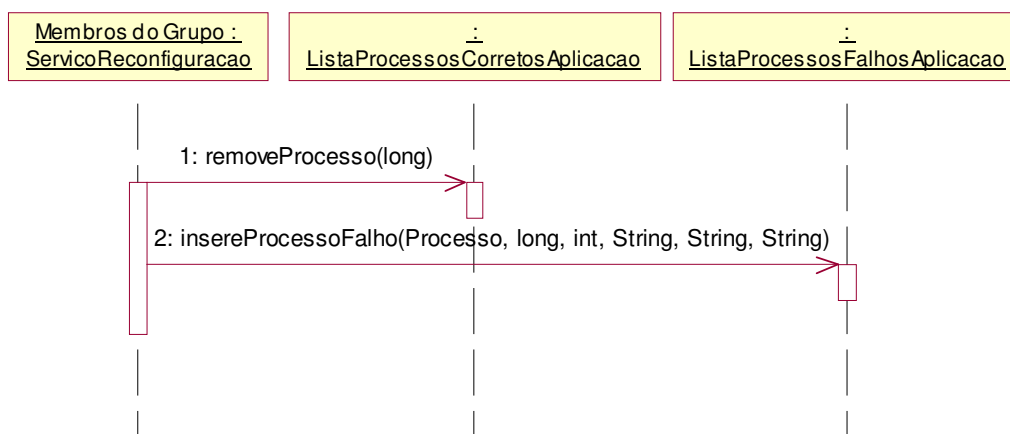


Figura 4.35 - Diagrama : Retirada do processo falho do objeto ListaProcessosCorretosAplicacao e inserção no objeto ListaProcessosFalhosAplicação

4.5.3. Falha do Módulo SDF Local de uma Máquina que Tenha Processos Monitorados em Estado Suspeito

O diagrama da figura 4.36 apresenta as atividades executadas quando não é possível acessar o objeto `ServicoDFAgenteObjetos`, através da chamada a operação `investigaSOProcesso()`, a fim de solicitar a este que investigue um processo que não responde solicitações remotas. Quando o objeto `ServicoDFGerente` recebe o resultado desta investigação, `recebeResultadoInvestigaProcesso()`, ele dispara uma chamada ao objeto `ServicoReconfiguracao` para que este atualize a informação de falha de um módulo SDF, `atualizaEstadoMaquinaServicoDiagnosticoInativo()`. O coordenador do grupo então inicia o protocolo de acordo para retirada dos processos da máquina que está com o módulo SDF inativo. As atividades deste protocolo de acordo são descritos mais adiante no item 4.5.5.

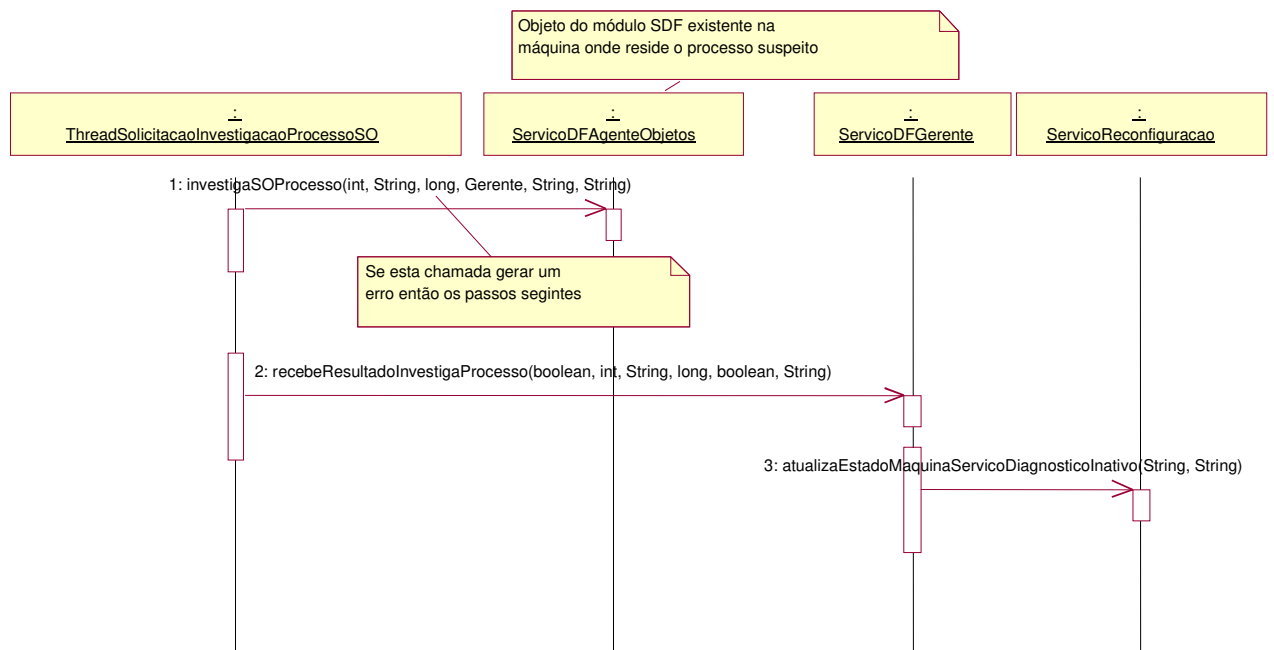


Figura 4.36 - Diagrama : Detecção de falha do módulo SDF de uma máquina com processo suspeito

4.5.4. Falha na Máquina que Contém Processos Monitorados

O diagrama da figura 4.37 mostra as atividades executadas quando é detectado que a máquina de um processo falho não está disponível. O objeto `ServicoDFAgenteConexao` reporta a falha ao objeto `ServicoDFGerente` local, através do método `acionaGerenteInvestigacaoProcesso()`, e este atualiza o estado de todos os processos residentes nesta máquina como suspeito, `atualizaSeProcessoSuspeito()`. Em seguida, o objeto `ServicoReconfiguracao` é informado para retirar os processos da máquina do grupo, `retiraProcessosDaMaquina()`. O objeto `ServicoReconfiguracao` coloca o endereço IP em uma tabela de máquinas falha através da chamada, `adicionaIpTabela()`, e se for o coordenador inicia o protocolo de acordo. O objeto `ServicoReconfiguracao` que não é coordenador testa o objeto coordenador para certificar que este está ativo, `verificacaoDoCoordenador()`.

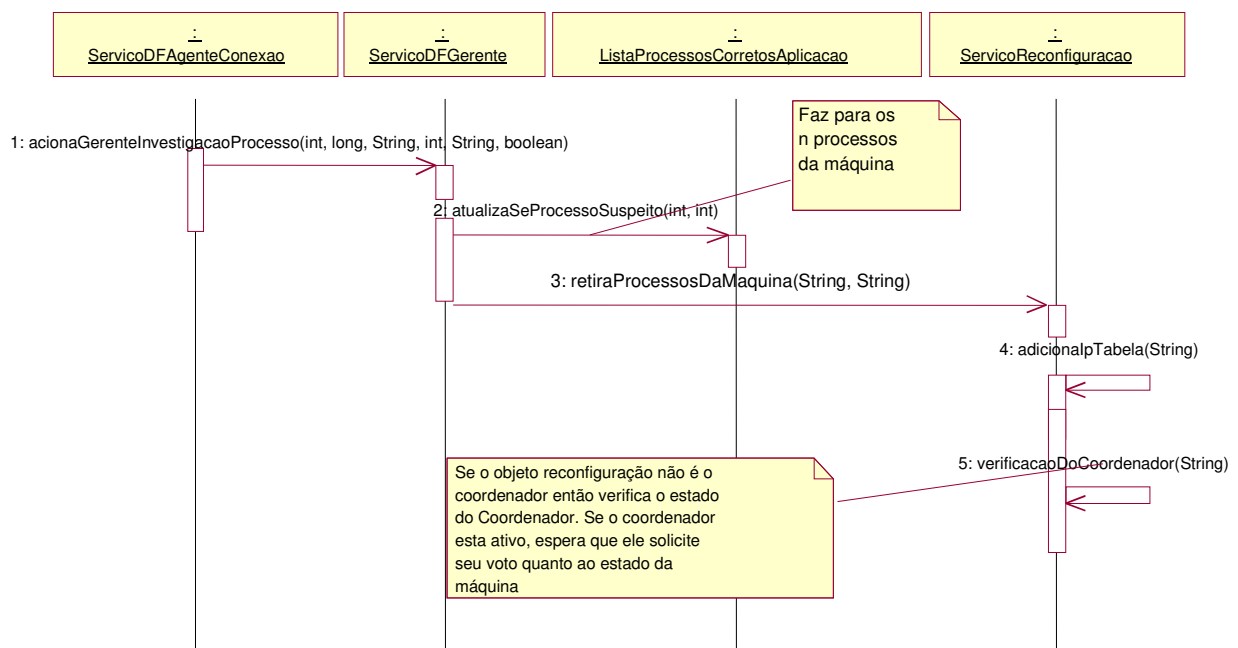


Figura 4.37 - Diagrama : Detecção de indisponibilidade de uma máquina que tem processos monitorados.

Em alguns casos de falha de máquina, o que ocorre, na verdade, é a interrupção temporária na comunicação da máquina com o servidor de nomes. Quando isto acontece é possível o retorno da máquina falha ao grupo. A seguir mostraremos as atividades necessárias para que uma máquina considerada como falha retorne ao grupo.

4.5.4.1. Retorno ao Grupo de uma Máquina Falha

O diagrama de seqüência apresentada na figura 4.38 mostra as ações tomadas por uma máquina que fica sem acesso ao servidor de nomes. A seqüência de atividades na figura 4.39 apresenta as ações executadas quando há um retorno desta máquina ao grupo após o restabelecimento da comunicação com o servidor de nomes.

No primeiro diagrama (figura 4.38) o objeto *ServicoReconfiguracao* percebe que está sem comunicação com o servidor de nomes e, por conta disso, solicita ao objeto *ServicoDFGerente* que interrompa a monitoria dos processos, *paraMonitoria()*. A partir deste momento, ele começa a tomar as ações necessárias para contornar esta situação. Solicita que os processos locais da máquina sejam inseridos no objeto *ListaExcluidosParticionamento*, *insereProcesso()*, e aciona o objeto *ServicoDFGerente*, *comecaMonitoriaProcessosLocalEServidorNomes()*, para que este inicie uma monitoria à máquina do servidor de nomes. O objeto *ServicoDFGerente* cria uma *thread* de monitoria, *ThreadMonitoriaServidorNomes()*, que fica constantemente tentando contato com a máquina do servidor nomes, *doPing()*. Quando a comunicação é restabelecida, o objeto *ServicoReconfiguracao* é avisado, *finalizaParticionamento()*.

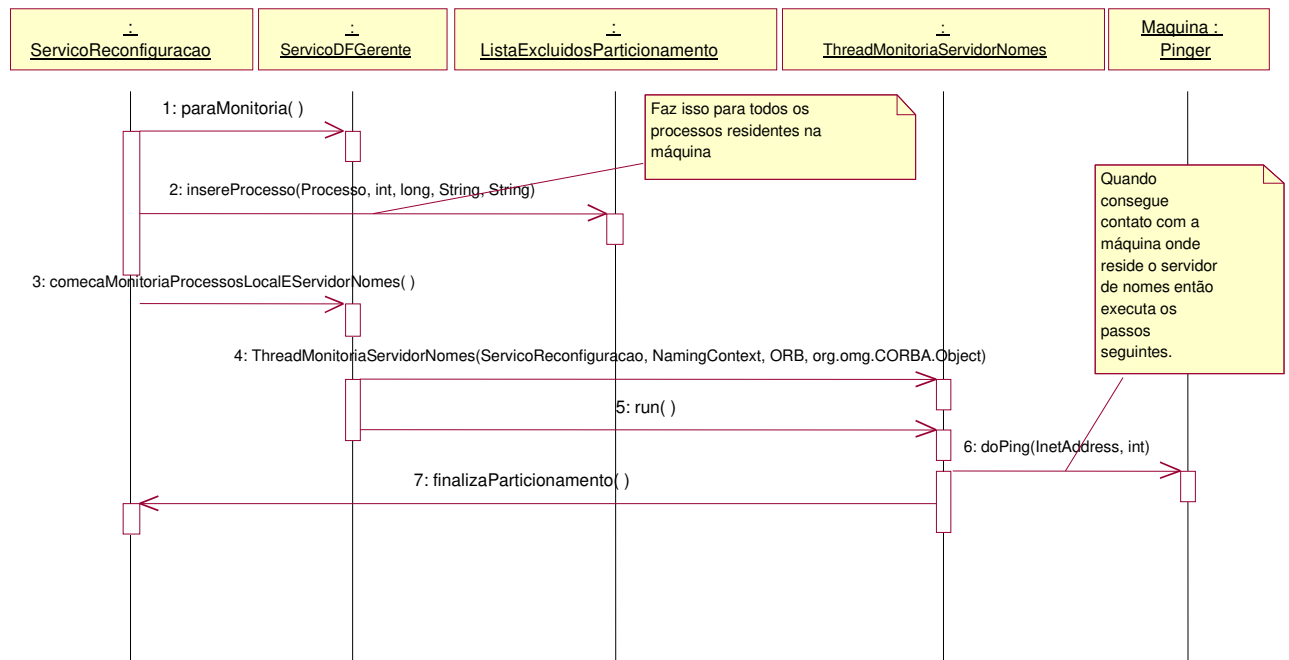


Figura 4.38 - Diagrama : Ações tomadas quando uma máquina perceber que esta sem contato com o Servidor de Nomes

A seqüência na figura 4.39 mostra as atividades executadas quando há um retorno de uma máquina ao grupo depois que a comunicação com o servidor de nomes é restabelecida. O objeto ServicoReconfiguracao solicita as informações de estado dos membros do grupo. Isto envolve o histórico das máquinas, *retornaHistoricoMaquina()*, os processos corretos, *informaProcessosCorretos()*, e os processos falhos, *informaProcessosFalhos()*. Solicita também ao coordenador do grupo que sua máquina seja retirada da lista de máquinas indisponíveis, *solicitaRetiradaDoParticionamento()*. Em seguida, o objeto ServicoReconfiguracao recebe as informações sobre os processos locais que estavam no objeto ListaExcluidosParticionamento através de chamadas a este objeto, *retornaTabela()* até *retornaTabelaFalha()*, criando uma *thread* para coordenar a inserção de cada processo no grupo novamente, *ThreadInsereObjetoParaGerenciamento()*. Esta *thread* acionará o processo, *insereNovamenteObjetoParaGerenciamento()*, que deve, em seguida, solicitar sua inclusão no grupo, *recebeSolicitacaoInsercaoGrupo()*. No final da seqüência os processos devem ser retirados do objeto ListaExcluidosParticionamento, *removeTodos()*.

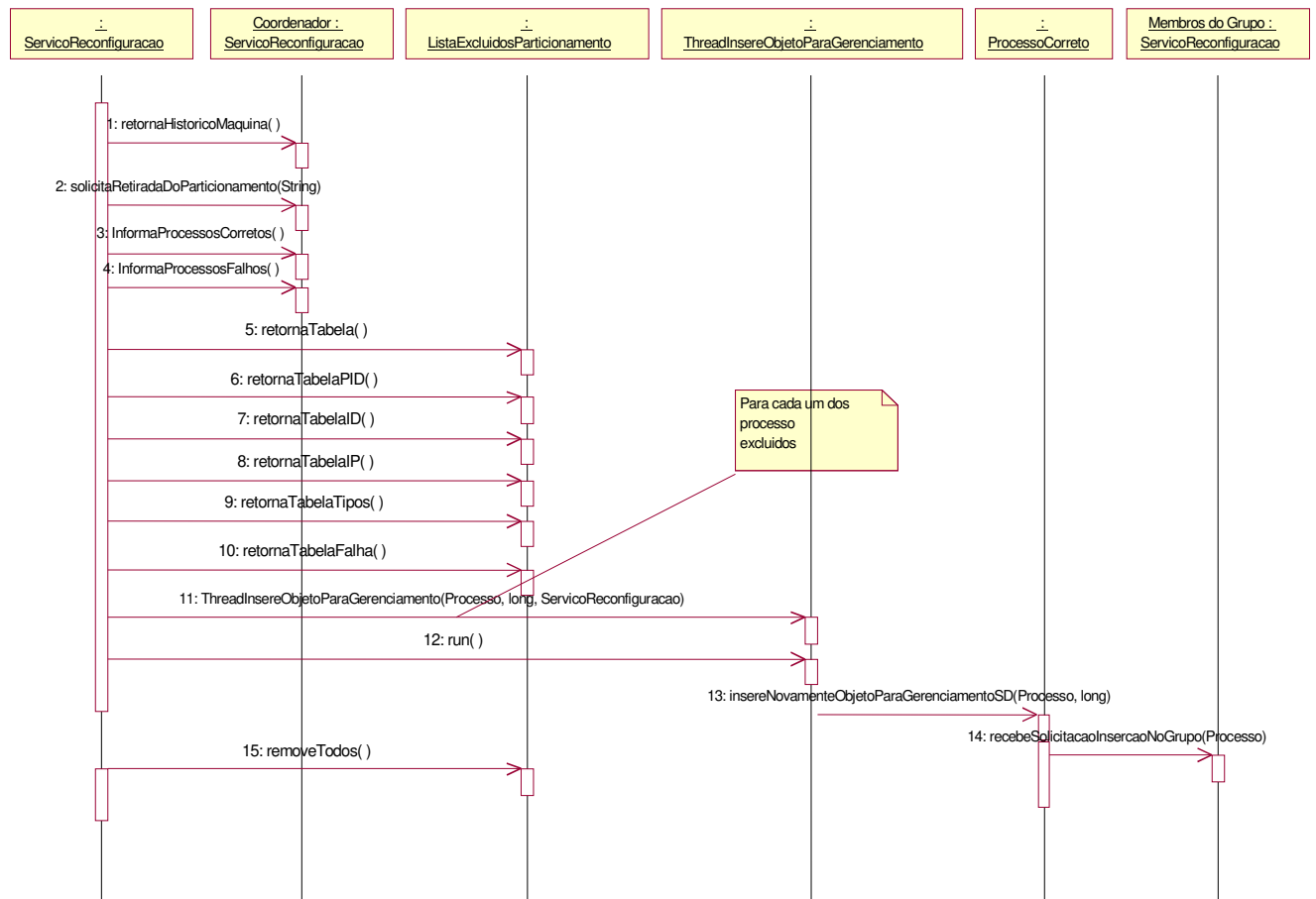


Figura 4.39 - Diagrama : Atividades executadas quando uma máquina volta ao grupo depois de ter sido particionada.

4.5.5. Diagramas de Seqüência do Protocolo de Acordo do SDF

O protocolo de acordo para retirada de máquinas falhas trabalha em duas fases, como foi visto no capítulo 3 desta dissertação. A primeira fase é mostrada no diagrama da figura 4.40, e a segunda fase está representada na figura 4.41.

Na primeira fase, o coordenador conta quantos módulos SDF ativos existem, *contaModulos()*, inicializa uma *thread* de temporização para analisar os votos, *TemporizadorCoordenador()*, e uma *thread* de solicitação de votos, *ThreadSolicitaVotos()*. A *thread* de solicitacao de votos inicializa uma *thread*, *ThreadVotos()*, para cada objeto *ServicoRe configuracao* a fim de solicitar que este envie o voto sobre o estado de uma máquina suspeita, *enviarVotoParaCoordenador()*. O

coordenador do grupo recebe os votos através do método *recebeResultado()*. Quando o objeto *TemporizadorCoordenador* termina seu *timeout* de espera pelos votos, ele aciona o coordenador para que este analise os votos recebidos, *analisaVotos()*.

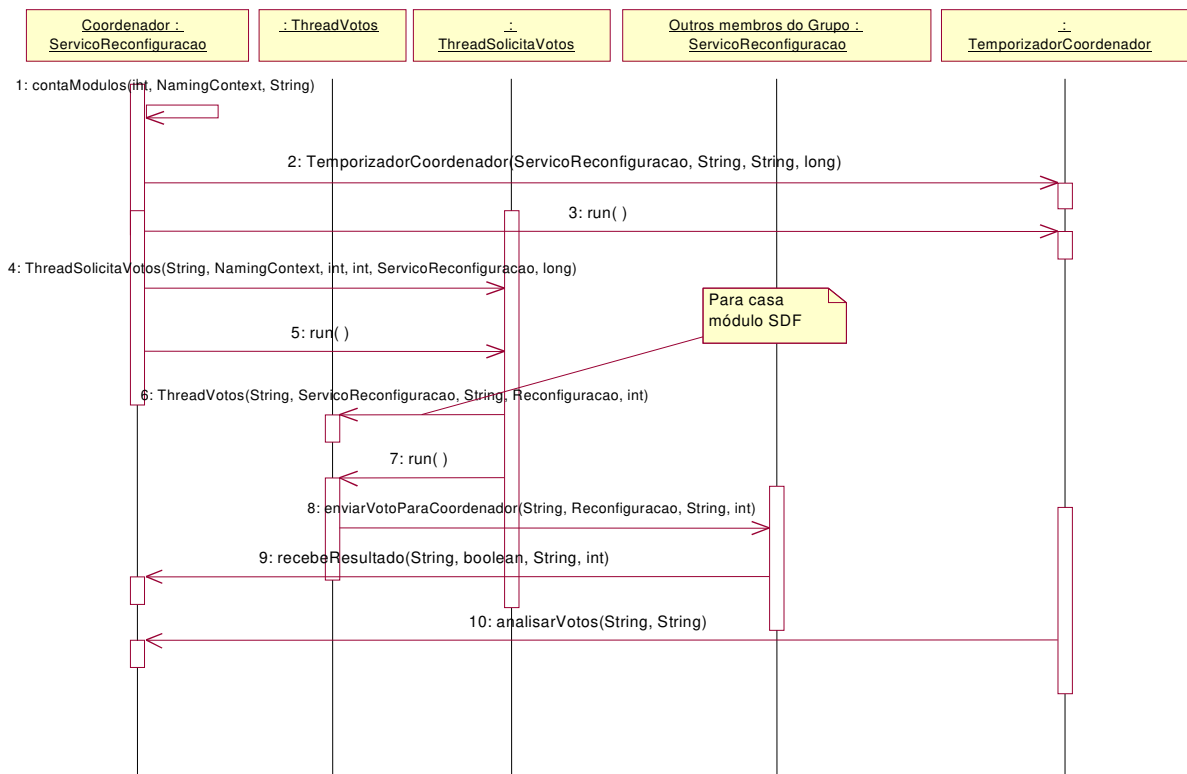


Figura 4.40 - Diagrama : Primeira fase do protocolo de acordo para retirada do grupo dos processos de uma máquina falha

Na segunda fase do protocolo, se a máquina suspeita é considerada falha por todos os módulos ativos, uma *thread* de retirada de processos é criada pelo coordenador do grupo, *ThreadRetiraProcessosMaquinaFalha()*. Esta *thread* de retirada inicializa uma *thread*, *ThreadRetiraProcessoGrupo()*, para cada ocorrência de processo da máquina falha nos objetos *ServicoRe configuracao*. Em seguida é solicitado que o processo seja retirado do grupo, *retiraProcessoGrupo()*. O objeto *ServicoRe configuracao* ao receber esta solicitação, remove o processo do objeto *ListaProcessoCorretosAplicacao*, *removeProcesso()* e insere no objeto *ListaProcessosFalhosAplicacao*, *insereProcessoFalhos()*, como pode ser visto no diagrama da figura 4.41.

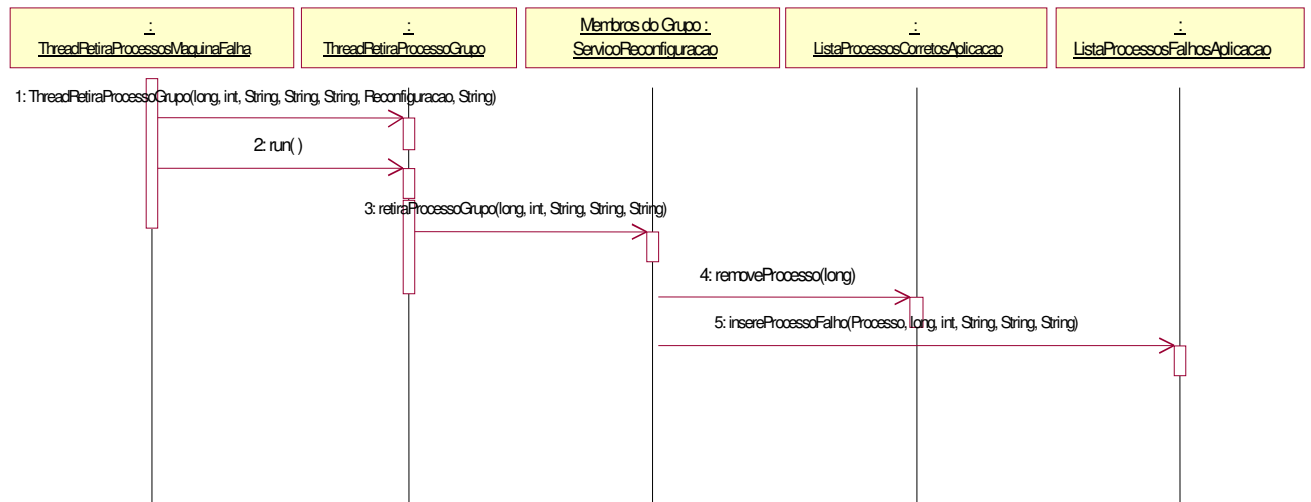


Figura 4.41 - Diagrama : Segunda fase do protocolo de acordo quando há unanimidade sobre a falha de uma máquina

4.6. Referência aos Objetos SDF no Servidor de Nomes

Como visto anteriormente, em todas as máquinas onde existem processos monitorados existe um módulo do SDF. Na nossa implementação, estes módulos utilizam o servidor de nomes do Visibroker para obterem as referências aos objetos SDF em outras máquinas.

No momento da inicialização de um modulo SDF em uma máquina, os objetos das classes ServicoReconfiguração, ServicoDFGerente e ServicoAgenteObjetos são cadastrados no servidor de nomes. Assim, um módulo SDF existente na máquina com endereço IP x.x.x.x, terá no servidor de nomes as entradas ID_SR_x.x.x.x, ID_SDFG_x.x.x.x e ID_SDFAO_x.x.x.x referentes aos objetos ServicoReconfiguração, ServicoDFGerente e ServicoAgenteObjetos. O parâmetro ID existente na referência ao objeto é necessário pois é possível ter mais de um módulo do SDF na mesma máquina. Esta situação acontece quando é necessário monitorar processos de uma mesma máquina pertencentes a aplicações diferentes.

Se existir um módulo SDF na máquina y.y.y.y então existirá também as seguintes entradas no servidor de nomes ID_SR_y.y.y.y, ID_SDFG_y.y.y.y e ID_SDFAO_y.y.y.y, referentes

aos objetos do módulo nesta máquina. Neste caso, se o módulo SDF da máquina y.y.y.y não for o primeiro módulo vinculado à aplicação, ele solicitará ao módulo que atua como coordenador que lhe mande os objetos ListaProcessosCorretosAplicacao e ListaProcessosFalhosAplicacao, ou seja o estado de todos os processos monitorados pelo serviço SDF.

A figura 4.42 mostra a organização das referências dos objetos SDF no servidor de nomes, considerando o exemplo hipotético de máquinas com IP x.x.x.x e y.y.y.y que tenham o módulo do serviço ativo para uma aplicação cujo identificador (ID) é igual a 2.

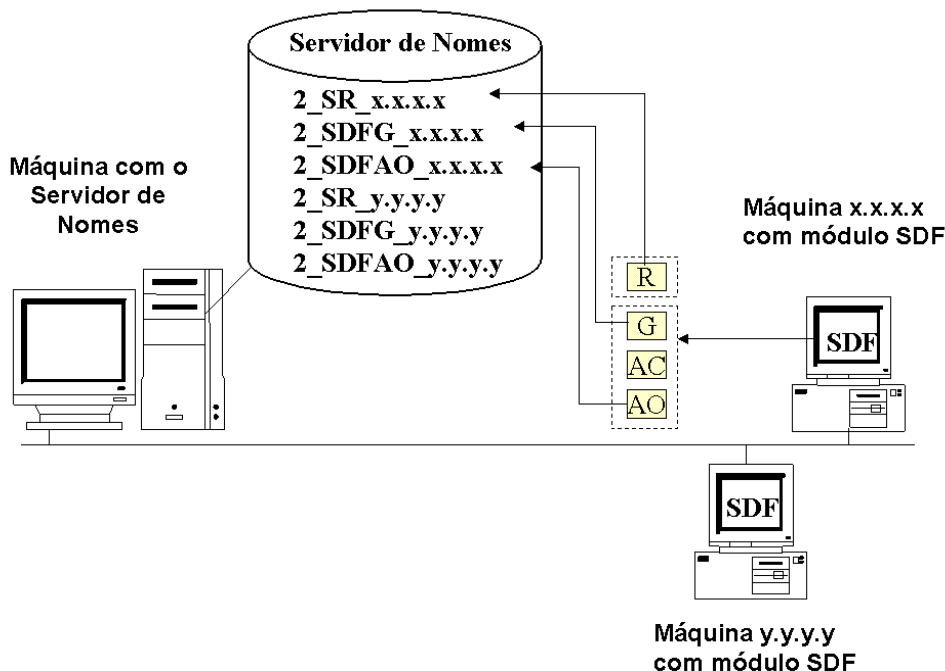


Figura 4.42 - Organização das referências a objetos do SDF no servidor de nomes

No exemplo da figura 4.42, se o objeto ServicoGerente da máquina y.y.y.y precisar se comunicar com o objeto ServicoDFAgenteObjetos da máquina x.x.x.x, ele consultará o servidor de nomes passando como parâmetro 2_SDFAO_x.x.x.x e receberá de volta a referência a este objeto. Com esta referência, para estabelecer uma comunicação, é só fazer a chamada sobre a plataforma CORBA. No item seguinte apresentaremos como

implementamos os componentes do SDF como objetos CORBA e como acontece a comunicação entre estes objetos sobre esta plataforma.

4.7. Comunicação Entre os Objetos do SDF Sobre a Plataforma CORBA

O CORBA Object Request Broker (ORB) estabelece comunicação entre objetos em um ambiente distribuído permitindo que um objeto cliente invoque transparentemente métodos em um objeto servidor que pode estar na mesma máquina ou então em outra máquina da rede.

Dentro do ambiente CORBA um objeto pode atuar como servidor de solicitações e, em outro momento, atuar como cliente fazendo requisições a outro objeto.

Para que um objeto responda a requisições basta que sejam definidas em IDL (*linguagem de definição de interfaces*) as operações que poderão ser atendidas via plataforma CORBA. Em seguida, o arquivo IDL passa por um compilador de uma linguagem de programação com mapeamento IDL afim de serem geradas as classes necessárias à comunicação. Da compilação surgem interfaces para cada objeto que teve suas operações descritas na IDL, e classes que atuam como Stubs e Skeleton numa aplicação CORBA.

Para fazer uma requisição, o objeto cliente precisa ter acesso à referência do objeto que pode ser obtida através de servidores de nomes ou então por solicitação a outros objetos. Um vez tendo a referência, a requisição de operações pode acontecer do lado do objeto cliente através das Stubs que foram definidas para o objeto servidor. Estas Stubs definem como os objetos clientes devem requisitar as operações aos objetos servidores.

Do lado do objeto servidor, o ORB, junto com o adaptador de objetos, utiliza o Skeleton para invocar os métodos que foram definidos na IDL do objeto. O Skeleton de um objeto é a classe, gerada na compilação da IDL, que implementa a interface do objeto (também gerada na compilação da IDL). Este Skeleton deve ser herdado pela implementação do objeto servidor.

Na implementação do SDF, a linguagem de programação com mapeamento IDL utilizada foi o Java e o compilador o *idlToJava*, que faz parte do Visibroker 3.4.

Foram definidos em IDL operações para os objetos *reconfiguração*, *gerente*, *agente de objetos*, *lista de processos corretos e processo*. Estes objetos respondem a um conjunto de operações que podem estar sendo solicitadas por objetos existentes em outras máquinas da rede.

Para os objetos clientes, a invocação de operações sobre um objeto servidor ocorre como se esta fosse uma rotina local. O ORB, através do adaptador de objetos, redireciona de forma transparente a chamada para o objeto servidor.

Os objetos SDF obtêm as referências a objetos remotos no servidor de nomes do Visibroker 3.4 ou então de outros objetos remotos.

Na figura 4.43 podemos ver a IDL que foi gerada para os objetos do SDF que precisam ser invocados remotamente. Por uma questão de simplificação colocamos na IDL da figura apenas as principais operações dos objetos. Esta IDL, quando compilada pelo *idlToJava*, gera interfaces Java para cada objeto definido, a saber, uma interface *Reconfiguracao*, uma *Gerente*, uma *AgenteObjetos*, uma *ListaProcessos* e por último, uma interface *Processo*.

Da compilação de uma IDL, além das interfaces, surgem as classes que atuarão como Stubs e Skeleton para a invocação de métodos entre os objetos. Outras classes auxiliares também são geradas além das classes Stubs e Skeleton, no processo de compilação pelo *idlToJava*.

```

module Pack_Servico {

    interface Processo {
        readonly attribute long Id;
        readonly attribute long long PID;
        readonly attribute string Ip;
        readonly attribute string tipoObjeto;
        boolean respondeMonitoria();
        void insereObjetoParaGerenciamentoSD(in Processo p, in long long l);
        boolean insereNovamenteObjetoParaGerenciamentoSD(in Processo PC, in long long pid);
    };
    typedef sequence<long>SequenciaId;
    typedef sequence<Processo>Sequencia;
    typedef sequence<long long>SequenciaPID;
    typedef sequence<string>SequenciaPMaquinas;
    typedef sequence<string>SequenciaTiposObjetos;
    typedef sequence<string>SequenciaFalha;
    typedef sequence<boolean>SequenciaEstados;
    interface ListaProcessos {
        ListaProcessos retornaProcessos();
        readonly attribute long chave;
        Sequencia retornaTabela();
        SequenciaId retornaTabelaID();
        SequenciaId retornaTabelaSuspeito();
        SequenciaPID retornaTabelaPID();
        SequenciaPMaquinas retornaTabelaIP();
        SequenciaTiposObjetos retornaTabelaTipos();
        SequenciaFalha retornaTabelaFalha();
        SequenciaEstados retornaTabelaEstadoMaquina();
    };

    interface Gerente{
        string retornaValorTabelaQoSAtual(in string identificaDePara);
        string retornaValorTabelaQoSValorAtual(in string identificaDePara);
        void recebeResultadoInvestigaProcesso(in boolean result, in long idProcesso, in string
        ipMaquina, in long long valorPID, in boolean problemaMaquinaServicos, in string TipoObjeto);
    };
    interface Reconfiguracao {
        void atualizaNovoCoordenador(in string ipNovoCoordenador, in string falha);
        void verificarEstadoSR();
        boolean receberVotosProximoCoord(in string propostaCoord, in string ipLocal);
        void informeVotoProximoCoordenador(in Reconfiguracao coord);
        void retiraProcessos(in string guardaIpCoordAnt, in string falha);
        void enviarVotoParaCoordenador(in string ipMaquina, in Reconfiguracao Coord, in string
        nomeAplicacao, in long numeroRodada);
        boolean recebeResultado(in string ipMaquina, in boolean resultado, in string ipLoc, in long
        numeroRodada);
        void retiraProcessoGrupo(in long long PID, in long id, in string ip, in string falha, in string
        tipoObjeto);
        void solicitaRetiradaDoParticionamento(in string ipMaquina);
        void recebeSolicitacaoInsercaoNoGrupo(in Processo p);
        void atualizaHistoricoMaquina(in string ipLocal, in string estado, in long numHist);
        void atualizaNumeroEstadoReconfiguracao(in long estado, in long numRodada);
    };

    interface AgenteObjetos{
        void investigaSOProcesso(in long ID, in string IP, in long long PID, in Gerente G, in string
        TipoObjeto, in string ipSolicitador);
    };
}

```

Figura 4.43 - IDL do SDF com apenas algumas das operações principais

Como exemplo, mostramos na figura 4.44 a IDL do objeto reconfiguração da arquitetura SDF, a interface gerada a partir desta IDL e as classes Stubs e Skeleton que implementam a interface Java Reconfiguracao. Pode-se ver também, nesta figura, a classe ServicoReconfiguracao, que faz parte das classes do SDF, e que herda da classe Skeleton, _ReconfiguracaoImpBase. Desta forma, os objetos instanciados da classe ServicoReconfiguracao podem receber requisições aos métodos que foram especificados na interface IDL para o objeto reconfiguracao.

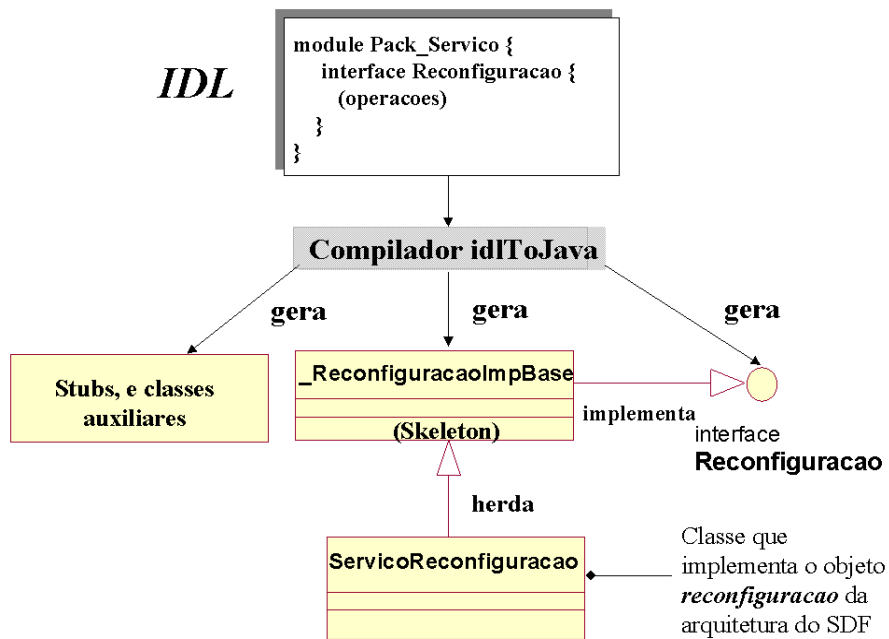


Figura 4.44 - Classe ServicoReconfiguracao herda métodos de classe originadas a partir de uma IDL

Na figura 4.45 podemos ver as classes do SDF, que respondem a solicitações remotas de outros objetos, com os seus respectivos Skeleton gerados na compilação da IDL apresentada na figura 4.43.

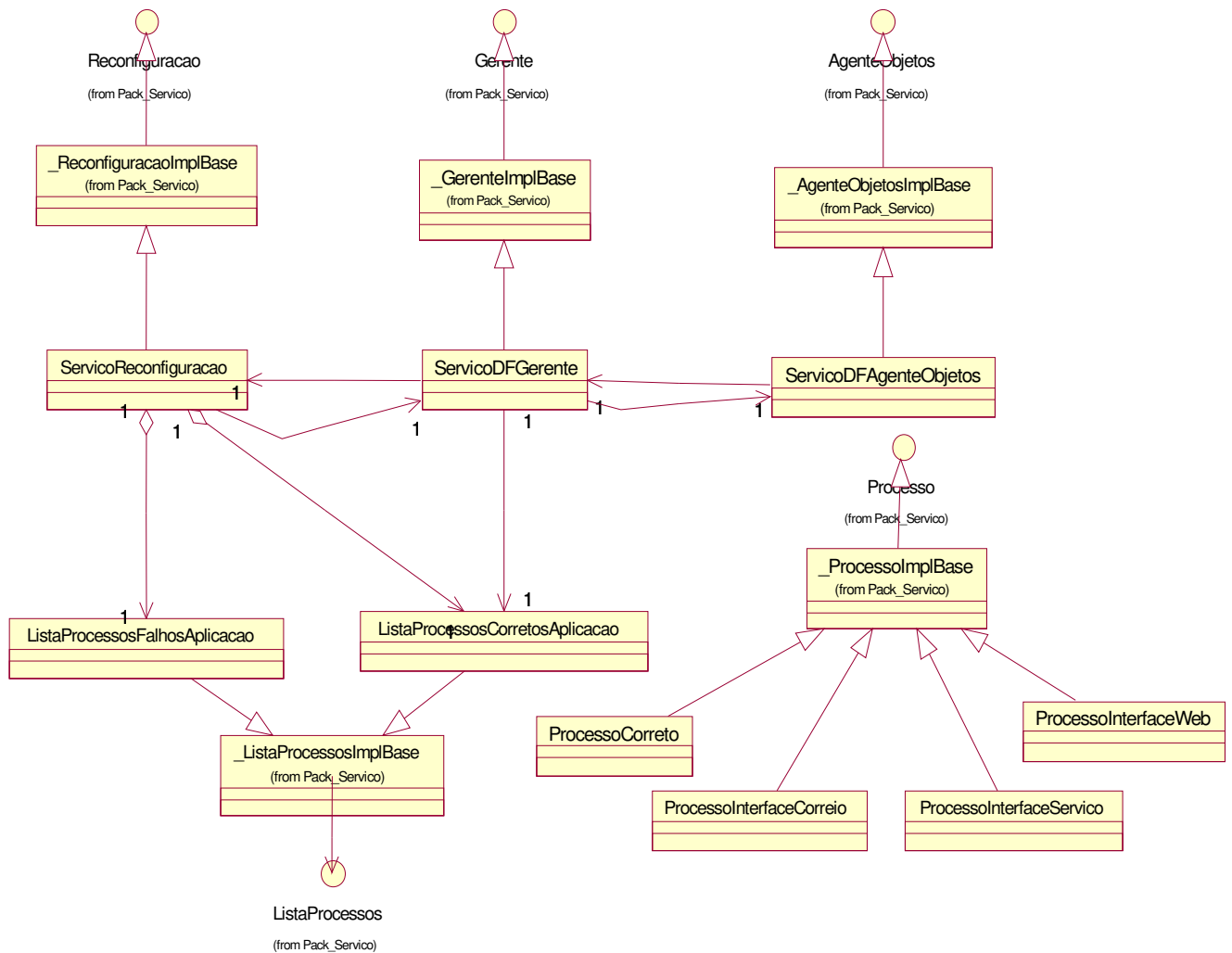


Figura 4.45 - Classes do SDF com seus respectivos Skeleton gerados na compilação da IDL

Como pode ser visto no diagrama da figura 4.45, a classe ServicoDFGerente do SDF herda da classe Skeleton _GerenteImplBase, que implementa a interface Gerente criada na compilação pelo idlToJava. A classe ServicoDFAgenteObjetos herda da classe Skeleton _AgenteObjetosImplBase, que implementa a interface AgenteObjetos, enquanto que as classes ListaProcessosFalhosAplicação e ListaProcessosCorretosAplicação herdam da classe Skeleton _ListaProcessosImplBase que por sua vez, implementa a interface ListaProcessos. A figura mostra também as classes de Processos (ProcessosCorretos, ProcessoInterfaceCorreto, ProcessoInterfaceServico e ProcessoInterfaceWeb) herdam da classe Skeleton _ProcessoImplBase, que implementa a interface Processo.

4.8. Resumo dos Aspectos Principais Apresentados no Capítulo

Neste capítulo mostramos as classes desenvolvidas para implementação da arquitetura apresentada no capítulo anterior. Inicialmente descrevemos as classes do núcleo básico do SDF, com suas principais operações. Em seguida, mostramos as classes que auxiliam o núcleo do SDF a executar as atividades de detecção e diagnóstico. Muitas destas classes são *threads* que executam atividades específicas para as classes principais do serviço. Apresentamos as classes do Visualizador que permite aos usuários acompanhar o estado dos processos monitorados.

Definimos interfaces para possibilitar monitoria pelo SDF. Uma interface genérica, chamada de *ObjetosGerenciados*, foi definida com o objetivo de fornecer um padrão de monitoria independente do tipo de objeto. Uma interface específica para processos, chamada de *ObjetosSistemas*, foi proposta e a monitoria sobre objetos desta natureza foi implementada. Uma interface chamada de *ObjetosRedes* foi definida para monitorar recursos de rede, tais como hubs e switch, mas não foi implementada por estar fora do escopo do trabalho desta dissertação.

As classes criadas para representar os diferentes tipos de processos investigados pelo SDF também foram mostradas neste capítulo. Apresentamos como foi implementada a noção de processo interface que tem por objetivo possibilitar a serviços que não implementam a interface *ObjetosSistemas* serem monitorados pelo SDF.

Apresentamos diagramas de seqüência que representam as atividades do SDF para detecção de falhas nos processos. Dividimos os diagramas pelos tipos diferentes de falhas que são detectadas pelo serviço. Apresentamos também os diagramas de seqüência que representam as etapas do protocolo de acordo para retirada dos processos de uma máquina falha ou com módulo SDF em estado falho.

Descrevemos também como implementamos as referências aos objetos SDF no servidor de nomes e como estes objetos se comunicam sobre a plataforma CORBA.

Capítulo 5

Testes e avaliação de desempenho

Neste capítulo apresentaremos o resultado dos testes executados com o Serviço de Diagnóstico. As conclusões sobre o desempenho do serviço são apresentadas no final do capítulo.

5. Testes e Avaliação de Desempenho

Neste capítulo serão descritos os testes realizados com o serviço de diagnóstico e as conclusões resultantes destes testes.

Os resultados dos testes foram encontrados utilizando a estatística não-paramétrica [FM96]. Este tipo de estatística é aplicável a pequenas amostras e é adequada para análise de dados qualitativos. Assumimos que os valores utilizados nas análises estão sujeitos a um nível de significância de 5%, ou seja, um valor de erro tolerável de 5%.

O capítulo está dividido da seguinte forma: no item 5.1 apresentamos o ambiente onde os testes foram realizados; no item 5.2 definimos o que estamos medindo, ou seja, as métricas que foram utilizadas nos testes; no item 5.3 descrevemos os experimentos feitos e identificamos os objetivos dos testes; no item 5.4 apresentamos as amostras coletadas para utilização nos testes; no item 5.5 mostramos os resultados; e finalizamos no item 5.6 com as conclusões sobre o comportamento do serviço de diagnóstico.

5.1. Ambiente dos Experimentos

Os testes foram realizados em máquinas PCs 300 Mhz, com 64 MB de memória, conectadas por uma rede Ethernet 10 Mbps. Estas máquinas tinham como sistema operacional o software Windows NT Server 4.0. A plataforma CORBA foi utilizada através do uso do produto Visibroker 3.4 para Java, da Borland. Como visto no capítulo anterior, o serviço de diagnóstico foi implementado em Java, utilizando a versão 1.2.2 do JDK.

Os testes foram feitos com grupos de três, cinco e sete máquinas com módulos do serviço.

5.2. Definição das Métricas

Nossos testes visaram medir o tempo⁸ de reconfiguração do grupo, quando detectada a falha de processos. No caso de falhas individuais, este tempo compreende o período entre a detecção/identificação da falha, pelo módulo local, até que o último módulo⁹ do grupo confirme a retirada do processo falho da sua lista de processos corretos. No caso de falha de máquina, o tempo de reconfiguração começa a ser contabilizado após o acordo para retirada da máquina (isto é, de seus processos) do grupo e é finalizado quando o último módulo confirma esta retirada.

5.3. Descrição dos Experimentos e Objetivos

Para identificar o tempo de reconfiguração do grupo quando detectada a falha de processos monitorados, realizamos dois tipos distintos de testes.

No primeiro teste, forçamos a falha individual de um processo através da parada deste, uma falha do tipo *crash*, onde o processo pára e não realiza nenhuma computação depois disso. Isto foi feito por executar simultaneamente as teclas “*control c*” sobre a janela do Windows onde estava rodando o processo. Com este teste visamos identificar :

- 1) *se o aumento do número de processos no grupo impacta no tempo de retirada de um processo falho;*
- 2) *se o aumento do número de máquinas com módulos SDF impacta no tempo necessário para retirada de um processo falho do grupo.*

⁸ Todos os tempos descritos neste capítulo referem-se a valores em milissegundo, exceto quando dito, claramente, outra unidade de tempo.

⁹ A expressão último módulo não implica em uma ordem pré-estabelecida. Significa apenas o último módulo a enviar a confirmação de retirada do processo da lista de processos corretos.

No segundo teste, forçamos a falha de uma máquina com processos monitorados por desconectá-la fisicamente da rede. Desta forma, a máquina fica impossibilitada de emitir respostas às solicitações que estão sendo feitas, da mesma forma como acontece quando a máquina é desligada. O objetivo deste teste era identificar :

- 1) *Se o aumento do número de processos na máquina aumenta o tempo de reconfiguração do grupo em casos de falha desta máquina;*
- 2) *Se o aumento do número de máquinas com módulos SDF aumenta o tempo de reconfiguração para retirada dos processos de uma máquina que falhou.*

5.4. Amostras Para Utilização nos Testes

As tabelas 1 e 3 representam o conjunto das amostras coletadas para os testes sobre o serviço de diagnóstico. Subdividimos estas tabelas em três partes: *Amostra*, *Número do Caso* e *Estatísticas*.

A tabela 1 está relacionada ao primeiro teste. Nesta tabela, em *Amostra* apresentamos o número de máquinas com módulos do serviço de diagnóstico (**MAQ**), o número total de processos no grupo (**PROC**), e o número da amostra (**NUM**). Em *Número de Caso*, mostramos os valores encontrados nos testes. Para cada amostra repetimos o teste dez vezes, sob as mesmas condições (número do caso 1, 2, 3 até 10). Na parte de *Estatísticas* apresentamos a média, mediana e desvio padrão de cada amostra.

Os valores apresentados nesta tabela estão em milisegundos e representam tempos de reconfiguração do grupo decorrente da falha de um processo. Na Tabela 1 podemos perceber, por exemplo, que a amostra número (**NUM**) 3, refere-se a um grupo de 15 processos (**PROC**) distribuídos em 3 máquinas (**MAQ**). Nesta amostra, no número de caso 5, o tempo de reconfiguração encontrado para a retirada de um processo falho do grupo foi de 20 milisegundos.

Tabela 1. Síntese dos dados encontrados após os testes de retirada de processos individuais do grupo (todos os dados).

Amostra			Número do Caso										Estatísticas		
MAQ	PROC	NUM	1	2	3	4	5	6	7	8	9	10	Média	Mediana	Desv(A)
3	3	1	111	30	120	40	30	30	30	40	40	40	51.10	40.00	34.33
	7	2	120	100	51	30	50	20	40	40	31	40	52.20	40.00	32.17
	15	3	541	50	30	40	20	21	30	20	160	20	93.20	30.00	162.94
5	3	4	90	60	50	50	40	41	40	50	50	50	52.10	50.00	14.67
	7	5	150	71	60	30	50	30	50	30	30	50	55.10	50.00	36.33
	15	6	351	60	60	30	61	40	40	40	40	50	77.20	45.00	96.80
7	3	7	351	60	60	30	61	40	40	40	40	30	75.20	40.00	97.62
	7	8	120	31	170	40	50	30	70	111	60	40	72.20	55.00	46.61
	15	9	200	60	91	60	40	40	40	40	41	40	65.20	40.50	50.17

Ao começar uma nova amostra, reinicializamos o serviço e em seguida executamos a seqüência de testes cujos valores estão apresentados nos números de casos da Tabela 1. Examinando esta tabela, podemos perceber que os tempos encontrados no número de caso 1 são bem maiores do que os outros números de caso. Isto se dá pois estes tempos refletem as inicializações do SDF para a manipulação de processos que tenham falhado. Por isso, valores do número de caso 1 foram desconsiderados uma vez que refletem mais do que a retirada do processo falho do grupo, ou seja, estes valores englobam mais do que o tempo de reconfiguração do grupo que é o valor que nos interessa nos testes.

Excluimos também valores extremos da tabela 1, conhecidos como *outliers*, com $z^{10} \geq 1,96$. Estes tempos correspondem a 5 % das amostras encontradas, conforme pode ser visto na Tabela 1.1. Nesta tabela, o percentual acumulativo acima de 95%, corresponde a valores de tempo de reconfiguração maiores do que de 100 milisegundos. É possível ver também a freqüência dos valores de tempo de reconfiguração e o percentual que eles representam dentro do conjunto das amostras.

¹⁰ z representa o intervalo de confiança (definido neste trabalho em 95%) que é obtido pela multiplicação de z pelo desvio padrão amostral. $z = (x - \bar{x})/\sigma$

Tabela 1.1. Valores encontrados nas amostras da Tabela 1

Tempo de Reconfiguração (ms)	Frequência	Percentual	Percentual Acumulativo
20	4	4.9	4.9
21	1	1.2	6.2
30	15	18.5	24.7
31	2	2.5	27.2
40	25	30.9	58.0
41	2	2.5	60.5
50	12	14.8	75.3
51	1	1.2	76.5
60	9	11.1	87.7
61	2	2.5	90.1
70	1	1.2	91.4
71	1	1.2	92.6
91	1	1.2	93.8
100	1	1.2	95.1
111	1	1.2	96.3
120	1	1.2	97.5
160	1	1.2	98.8
170	1	1.2	100.0
Total	81	100	

A tabela 2 reflete as exclusões de $\mathbf{z} \geq 1,96$, representando apenas os dados de falhas individuais de processos que serão utilizados nos testes cujos resultados estão descritos no item 5.5 deste capítulo.

Tabela 2. Síntese dos dados encontrados após os teste de retirada de processos individuais do grupo (excluídos valores extremos).

Amostra			Número do Caso										Estatísticas		
MAQ	PROC	NUM	1	2	3	4	5	6	7	8	9	10	Média	Mediana	Desv(A)
3	3	1		30		40	30	30	30	40	40	40	35.00	35.00	5.35
	7	2			51	30	50	20	40	40	31	40	37.75	40.00	10.43
	15	3		50	30	40	20	21	30	20		20	28.88	25.50	11.15
5	3	4		60	50	50	40	41	40	50	50	50	47.89	50.00	6.53
	7	5		71	60	30	50	30	50	30	30	50	44.56	50.00	15.31
	15	6		60	60	30	61	40	40	40	40	50	46.78	40.00	11.33
7	3	7		60	60	30	61	40	40	40	40	30	44.56	40.00	12.52
	7	8		31		40	50	30	70		60	40	45.86	40.00	14.95
	15	9		60	91	60	40	40	40	40	41	40	50.22	40.00	17.54

A Tabela 3 está relacionada com o segundo tipo de teste que realizamos, falha de uma das máquinas do grupo com processos monitorados. Como na tabela 1, na tabela 3 em *Amostra* apresentamos o número de máquinas com módulos do serviço de diagnóstico (**MAQ**) e o número da amostra (**NUM**). No entanto, nesta tabela **PROC** refere-se à quantidade de processos existentes na máquina falha. Em número de caso, mostramos os valores encontrados nos testes. Para cada amostra repetimos o teste de falhar uma máquina do grupo dez vezes, sob as mesmas condições (*Número do Caso* 1, 2, 3 ... até 10). Na parte de *Estatísticas* apresentamos a média, mediana e desvio padrão de cada amostra. Todos os valores desta tabela representam tempos de reconfiguração do grupo, em milisegundos, decorrentes de falha de uma máquina com processos monitorados pelo SDF.

Por exemplo, olhando a tabela 3 podemos ver que a amostra número (**NUM**) 5, corresponde a um grupo com cinco máquinas (**MAQ**). Nesta amostra, falhamos uma máquina contendo três processos (**PROC**) monitorados. Repetimos este teste dez vezes encontrando os resultados existentes nos números de caso 1, 2, 3... 10. O número de caso 7 apresenta um tempo de reconfiguração do grupo de 80 milisegundos.

Tabela 3. Síntese dos dados encontrados após os testes de retirada de processos de uma máquina que falhou (todos os dados)

Amostra			Número do Caso										Estatísticas		
MAQ	PROC	NUM	1	2	3	4	5	6	7	8	9	10	Média	Mediana	Desv(A)
3	1	1	30	30	40	60	30	41	30	30	41	20	35.20	30.00	10.91
	3	2	50	50	40	40	40	51	40	40	50	40	44.10	40.00	5.30
	6	3	80	60	50	60	60	90	60	70	60	65	65.50	60.00	11.65
5	1	4	70	50	40	50	50	50	40	40	50	50	49.00	50.00	8.76
	3	5	80	70	80	80	80	101	80	90	80	80	82.10	80.00	8.14
	6	6	120	120	111	110	110	110	120	120	110	201	123.20	115.50	27.77

Também neste caso, excluímos da Tabela 3 valores extremos, os *outliers*, com $z \geq 1,96$, que representaram 5% dos valores das amostras encontradas conforme pode ser visto na Tabela 3.1. Nesta tabela, o percentual acumulativo acima de 95%, corresponde a valores de reconfiguração maiores do que de 120 milissegundos. Esta tabela apresenta também a frequência e o percentual dos tempos de reconfiguração encontrados nas amostras.

Tabela 3.1. Valores encontrados nas amostras da tabela 3

Tempo de Reconfiguração (ms)	Frequência	Percentual	Percentual Acumulativo
20	1	1.7	1.7
30	5	8.3	10.0
40	10	16.7	26.7
41	2	3.3	30.0
50	10	16.7	46.7
51	1	1.7	48.3
60	6	10.0	58.3
65	1	1.7	60.0
70	3	5.0	65.0
80	8	13.3	78.3
90	2	3.3	81.7
101	1	1.7	83.3
110	4	6.7	90.0
111	1	1.7	91.7
120	4	6.7	98.3
201	1	1.7	100.0
Total	60	100.0	

Depois de aplicadas as exclusões dos extremos temos como resultado a tabela 4. Esta tabela representa os dados de retirada de processos que residem em máquinas que falharam.

Tabela 4. Síntese dos dados encontrados após os testes de retirada de processos de uma máquina falha (excluídos os valores extremos)

Amostra			Número do Caso										Estatísticas		
MAQ	PROC	NUM	1	2	3	4	5	6	7	8	9	10	Média	Mediana	Desv(A)
3	1	1	30	30	40	60	30	41	30	30	41	20	35.20	30.00	10.91
	3	2	50	50	40	40	40	51	40	40	50	40	44.10	40.00	5.30
	6	3	80	60	50	60	60	90	60	70	60	65	65.50	60.00	11.65
5	1	4	70	50	40	50	50	50	40	40	50	50	49.00	50.00	8.76
	3	5	80	70	80	80	80	101	80	90	80	80	82.10	80.00	8.14
	6	6	120	120	111	110	110	110	120	120	110		114.56	111.00	5.17

No item seguinte veremos os resultados dos testes sobre os dados das tabelas 2 e 4. Estes testes visam responder as suposições apresentadas no item 5.3.

5.5. Resultados Estatísticos dos Testes

Utilizamos testes da estatística não-paramétrica sobre os valores encontrados nas tabelas 2 e 4. Escolhemos estes testes por serem aplicáveis a pequenas amostras ($n < 30$) e adequados para análise de dados qualitativos. A estatística não-paramétrica independe dos parâmetros populacionais e de suas respectivas estimativas [FM96]. Com estes testes não é necessário supor uma distribuição normal da variável na população. Dentre os testes não-paramétricos utilizamos os teste de *Kruskal-Wallis* e de *Mann-Whitney*.

O objetivo geral dos testes foi determinar se existe relação de igualdade de média entre as amostras coletadas, ou seja se o tempo de reconfiguração do grupo permanece constante com o aumento de processos ou módulos SDF. O teste de *Kruskal-Wallis* foi utilizado para decidir se n amostras independentes ($n > 2$), contendo os tempos de reconfiguração do

grupo em casos de falha, provinham de populações¹¹ com médias iguais. Quando provado que elas vinham de médias diferentes, aplicamos o teste de *Mann-Whitney*.

O teste de *Mann-Whitney* é usado para testar se duas amostras independentes foram retiradas de populações com médias iguais. É a alternativa ao teste paramétrico para igualdade de médias, já que neste caso não é exigida nenhuma consideração sobre as distribuições populacionais e suas variâncias. Utilizamos este teste para verificar, de duas em duas, quais as amostras que mostravam diferenças significativas nas médias, ou seja quais amostras que apresentaram diferenças nos valores médios de tempo de reconfiguração do grupo.

Tanto nos testes de *Kruskal-Wallis* como nos testes de *Mann-Whitney* uma diferença significativa é encontrada quando o valor calculado do **Asymp. Sig**¹² é menor do que 0.05. As fórmulas dos testes podem ser encontradas em [FM96, Spi93].

Para simplificar o trabalho de análise dos resultados utilizamos o software SPSS for Windows 8.0.0 [SNCKR98]. A seguir veremos os resultados encontrados em cada uma das categorias de falha testada (falhas de processos individuais e falhas de máquina com processos monitorados).

5.5.1. Retirada de Processos por Falhas Individuais

Os testes aplicados foram baseados na tabela 2 do item 5.4 e visam identificar:

- 1) *Se o aumento do número de processos no grupo impacta no tempo de retirada de um processo falho.*

Para isso aplicamos o teste de *Kruskal-Wallis* entre as amostras com quantidades diferentes de processos (três, sete e quinze) mas com a mesma quantidade de máquinas. Não encontramos diferenças significativas no valor médio de tempo de reconfiguração (valor do

¹¹ População, no contexto desta dissertação, refere-se a todos os tempos de reconfiguração do grupo por causa de uma falha.

¹² **Asymp.Sig** é a medida de nível de significância, ou seja o valor do erro tolerável de 5% (0.05).

Asymp. Sig.<0.05) com três máquinas (amostras número 1, 2 e 3 com Asymp. Sig = 0.142), com cinco máquinas (amostras número 4, 5 e 6 com Asymp. Sig. = 0.763) e com sete máquinas (amostras número 7, 8 e 9 com Asymp. Sig=0.699). Estes resultados foram obtidos através do uso do SPSS e as tabelas resultantes dos testes encontram-se no Anexo (tabelas A, B e C respectivamente).

Com os resultados encontrados, podemos dizer que o aumento do número de processos no grupo não impactou, de forma significativa, no tempo necessário para retirada de um processo falho, dadas as amostras coletadas. Isto nos leva a concluir que o serviço de diagnóstico se comporta de forma similar na retirada de um processo que falhou em um grupo com poucos processos ou com muitos (considerando a mesma quantidade de máquinas com módulos SDF). A explicação disto pode ser fornecida pela arquitetura do SDF. O módulo que detecta e identifica a falha solicita, através de *threads*, que os outros módulos retirem o processo falho da sua lista de processos corretos e, em cada módulo, esta ação é feita em memória com velocidade alta, não importando se o grupo tem poucos ou muitos processos.

- 2) *Se o aumento do número de máquinas com módulos SDF impacta no tempo necessário para a retirada de um processo falho do grupo.*

Neste caso, o teste de *Kruskal-Wallis* foi aplicado em amostras com diferentes quantidades de máquinas, mas com a mesma quantidade de processos. Testamos, inicialmente, as amostras número 1, 4 e 7, que se referem a um grupo de três processos em três, cinco e sete máquinas, respectivamente. O resultado deste teste indicou que existe uma diferença significativa nas médias (Asymp. Sig = 0.011), ou seja que houve uma diferença nas médias de tempo de reconfiguração quando comparadas as amostras. Em seguida, o mesmo teste foi repetido para as amostras número 2, 5 e 8, de um grupo composto por sete processos em três, cinco e sete máquinas e, neste caso, encontramos nos testes um resultado não significativo, Asymp. Sig = 0.669. Finalmente, para as amostras número 3, 6 e 9, de quinze processos em três, cinco e sete máquinas, com a aplicação dos testes de *Kruskal-Wallis* foi encontrado um resultado significativo de diferenças nas médias dos tempos de

reconfiguração, Asymp. Sig. = 0.008. As tabelas *D*, *E*, e *F*, encontradas no Anexo, contém os resultados dos testes de *Kruskal-Wallis* aplicados sobre estas amostras.

Como o teste de *Kruskal-Wallis* aplicado sobre as amostras número 1, 4 e 7 e sobre as amostras número 3, 6 e 9 apresentou uma diferença significativa das médias encontradas dos tempos de reconfiguração, foi aplicado sobre estes conjuntos de amostras o teste de *Mann-Whitney*. Analisando todos os resultados dos testes vimos que eles não foram concludentes quanto a se existe um impacto no tempo de retirada de um processo falho com o aumento do número de máquinas com módulos SDF. As tendências neste caso, devem aparecer em testes com um número maior de máquinas.

5.5.2. Retirada de Processos por Falha da Máquina Onde Este Reside

Neste item, os testes aplicados, foram baseados nos valores existentes na tabela 4 e têm por objetivo identificar:

- 1) *Se o aumento do número de processos na máquina aumenta o tempo de reconfiguração do grupo em casos de falha desta máquina.*

Para investigarmos esta hipótese, aplicamos o teste de *Kruskal-Wallis* entre as amostras com quantidades diferentes de processos na máquina falha (um, três e seis), mas em um grupo com a mesma quantidade de máquinas (três), respectivamente as amostras 1, 2 e 3 da tabela 4. O resultado deste teste encontra-se na tabela *G* do Anexo e indica uma diferença significativa nas médias dos tempos de reconfiguração das amostras (Asymp. Sig. = 0.00, ou seja, ≤ 0.05). Fizemos este mesmo teste para as amostras 4, 5 e 6 (cinco máquinas com módulos SDF e quantidades distintas de processos na máquina falha, respectivamente, um, três e seis). Também neste caso, os resultados apontam uma diferença significativa das médias (Asymp. Sig. = 0.00), como pode ser visto na tabela *H* do Anexo.

Como no teste de *Kruskal-Wallis* foi indicado que as amostras 1, 2 e 3 provinham de populações com médias diferentes aplicamos o teste de *Mann-Whitney*, para identificarmos

se, considerando de duas em duas amostras, os resultados encontrados indicam também uma diferença nas médias. O resultado indicou que entre as amostras 1 e 2 o de Asymp. Sig. = 0.051 (tabela I do Anexo), indica um valor limite do nível de significância, que é de 0.05. Entre as amostras 2, e 3 o resultado foi significativo com Asymp. Sig. =0.00 (tabela J do Anexo), e entre as amostras 1 e 3 o valor também foi bastante significativo com Asymp. Sig. = 0.00 (tabela L do Anexo).

Em seguida, considerando as amostras 4, 5 e 6, aplicamos o teste de *Mann-Whitney* entre as amostras 4 e 5 e entre as amostras 5 e 6. Nestes dois casos a diferença foi altamente significativa (Asymp. Sig. = 0.000), como pode ser visto nas tabelas M e N do Anexo.

Como resultado dos testes aplicados, podemos concluir que a quantidade de processos residentes em uma máquina falha impacta no tempo necessário para reconfiguração do grupo. Falhas de máquinas que têm muitos processos monitorados provocam no grupo reconfigurações mais demoradas do que as de máquinas com poucos processos monitorados. Isto se dá porque com mais processos a serem retirados, aumenta o tempo para o envio da confirmação de retirada e aumenta o tempo necessário pelo coordenador para processar todas estas mensagens. Como isto acontece em todos os módulos então o tempo total da reconfiguração do grupo aumenta.

2) *Se o aumento de números de máquinas com módulos SDF aumenta o tempo de reconfiguração para retirada dos processos de uma máquina falha.*

Com este teste visamos identificar se com o aumento de máquinas com módulos SDF aumenta o tempo de reconfiguração por falha de uma destas máquinas. Para isto aplicamos o teste de *Mann-Whitney* entre as amostras com quantidades diferentes de máquinas mas com o mesmo número de processos (amostras 1 e 4 ; 2 e 5 ; 3 e 6, da tabela 4). Não aplicamos o teste de *Kruskal-Wallis* pois as amostras foram retiradas de três e cinco máquinas ($n=2$). O resultado do teste de *Mann-Whitney* nas amostras 1 e 4 encontra-se na tabela O do Anexo e indica uma diferença significativa das médias (Asymp. Sig. =0.009), ou seja, houve um aumento do tempo necessário para a reconfiguração com o aumento do número de máquinas com módulos SDF. Da mesma forma, para as amostras 2 e 5 e para as

amostras 3 e 6, os resultados dos testes mostram diferenças significativas nas médias (Asymp. Sig = 0.000), como pode ser visto nas tabelas *P* e *Q* do Anexo.

Com estes testes foi possível concluir que há diferença nas médias dos tempos de reconfigurações, por causa de uma falha de máquina, com o aumento de máquinas com módulos SDF no grupo. Isso implica dizer que há um impacto no tempo necessário para a reconfiguração, a depender da quantidade de módulos SDF existentes no grupo. Este era um resultado esperado pois com o aumento de módulos SDF no grupo é necessária uma quantidade maior de mensagens para solicitar a retirada dos processos da máquina que falhou. Além disso, o tempo total para recebimento das confirmações de retirada dos processos de todos os módulos SDF aumenta na medida em que cresce o número de módulos.

5.6. Síntese dos Resultados Encontrados nos Testes

Neste capítulo foram descritos os testes executados sobre o serviço de diagnóstico de falhas e os resultados encontrados. Utilizando estatística não-paramétrica, analisamos as amostras coletadas e chegamos a algumas conclusões sobre o desempenho do SDF quando ocorrem falhas individuais de processos monitorados e falhas de máquinas com processos monitorados.

Inicialmente tratamos de falhas de processos individuais e do tempo necessário para reconfiguração do grupo neste caso. Pelos testes aplicados concluímos, com base nas amostras coletadas, que o aumento do número de processos no grupo não impacta de forma significativa no tempo de retirada de um processo falho. Os testes foram feitos para grupos com três, sete e quinze processos em três máquinas, cinco máquinas e sete máquinas. Este resultado era esperado visto que o módulo SDF solicita a retirada do processo falho aos outros módulos SDF através de threads independentes. Nos módulos, as listas de processos corretos e falhos são estruturas em memória que são percorridas muito rapidamente, o que não acarreta em maiores tempos com o aumento do número de processos no grupo.

No gráfico 5.1, podemos perceber que independente da quantidade de máquinas com módulos SDF (3 MAQ, 5 MAQ e 7 MAQ), o aumento do número de processos no grupo (3 PROC, 7 PROC e 15 PROC), não representou um aumento no tempo de reconfiguração, Tr (ms), quando ocorre a falha de um dos processos.

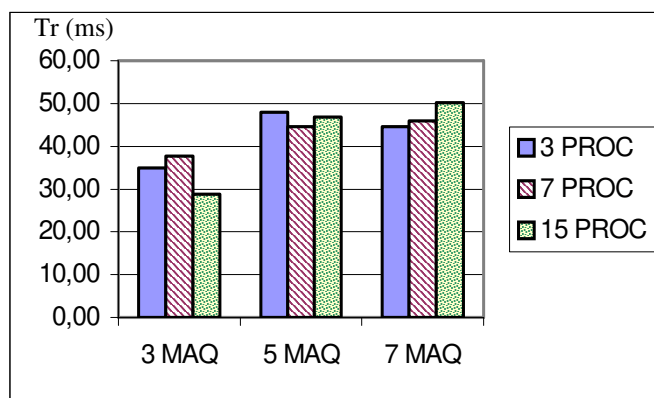


Figura 5.1 – Gráfico representando falhas de processos individuais em grupos com quantidades de processos diferentes (3, 7 e 15)

O outro teste, relativo a falhas individuais dos processos, visava verificar se o aumento do número de máquinas com módulos SDF no grupo impacta no tempo necessário para a retirada de um processo que falhou. Os resultados dos testes aplicados, neste caso, não foram concludentes. Possivelmente as tendências somente apareçam com um número maior de máquinas com módulos SDF no grupo.

Em seguida foi considerada a retirada do grupo de processos monitorados que se tornaram falhos por falha da máquina onde eles residiam. A primeira hipótese que averiguamos com os testes era se o aumento do número de processos na máquina aumentava o tempo de reconfiguração do grupo. Os resultados dos testes sobre as amostras indicaram que o número de processos residentes em uma máquina falha impacta no tempo necessário para reconfiguração do grupo. Esta tendência pode ser bem visualizada no gráfico apresentado na figura 5.2 cujos dados foram retirados da tabela 2. Neste gráfico vemos as médias encontradas nas reconfigurações de dois grupos; um com três e outro com cinco máquinas. Nos dois casos, é possível perceber que o tempo para reconfiguração do grupo, Tr (ms),

aumentou se a máquina falha tinha uma quantidade maior de processos (1 PROC, 3 PROC ou 6 PROC).

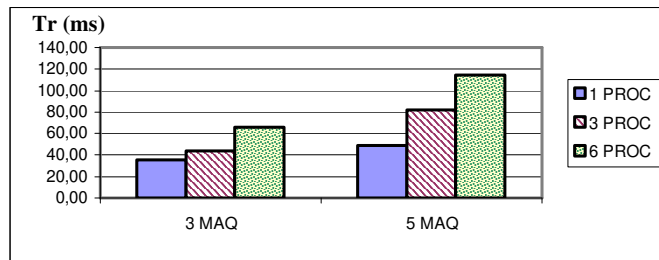


Figura 5.2– Gráfico das médias de tempo de reconfiguração quando ocorre falha de uma máquina – visão por número de máquinas no grupo.

A outra hipótese testada foi se o aumento do número de máquinas com módulos SDF impacta no tempo necessário para reconfiguração no caso de falha de uma destas máquinas. Os resultados dos testes apontaram também para um aumento no tempo de reconfiguração com o aumento do número de módulos SDF. Esta tendência pode ser visualizada no gráfico apresentado na figura 5.3 onde os dados da tabela 2 foram coletados e arrumados. Neste gráfico, é possível perceber que com o aumento do número de máquinas com módulos SDF no grupo (de 3 para 5) houve um aumento do tempo de reconfiguração, nos três casos testados (máquina falha com um processo, com três processos ou com seis processos).

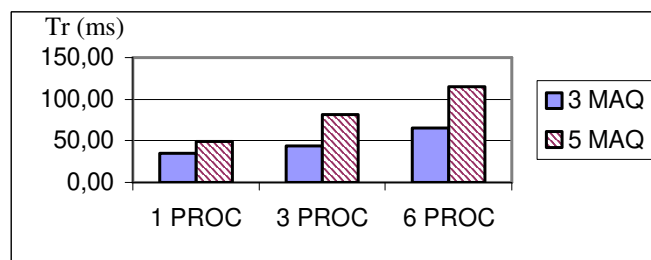


Figura 5.3 - Gráfico das médias de tempo de reconfiguração quando ocorre falha de uma máquina – visão por número de máquinas no grupo.

Isto se dá pois o tempo é medido no início da solicitação de reconfiguração, por parte do coordenador do grupo, até que o último módulo SDF informe a retirada da lista do último processo da máquina que falhou. Com uma quantidade maior de módulos este tempo tende a aumentar proporcionalmente.

Os resultados encontrados nos testes sugerem que a monitoria de aplicações com muitos processos em várias máquinas devem acontecer através de domínios de tolerância a falhas, conforme apresentado no capítulo 3. Isto diminuirá a quantidade de mensagens na rede para monitoria dos processos e, em redes onde não existe boa latência, pode diminuir também os tempos para reconfiguração do grupo. Neste caso, o acordo será restrito aos módulos SDF no domínio. Contudo, outros testes devem ser feitos para verificar o comportamento do SDF nestas condições.

Capítulo 6

Especificações de Tolerância a Falhas CORBA (FT-CORBA).

Neste capítulo apresentaremos o padrão FT-CORBA e relacionaremos os componentes do Serviço de Diagnóstico com os componentes propostos neste padrão.

6. Especificações de Tolerância a Falhas CORBA (FT-CORBA).

Atualmente muitas aplicações, que necessitam de um alto grau de confiabilidade, têm sido desenvolvidas utilizando o paradigma de orientação a objetos sobre uma plataforma CORBA. Como resultado disto, a OMG lançou em abril de 2000, um conjunto de especificações para introduzir aspectos de tolerância a falhas no padrão CORBA [OMG00]. Estas especificações contemplam aspectos de tolerância a falhas que não existiam no padrão CORBA original [OMG96] e visam o desenvolvimento de aplicações com um alto nível de confiabilidade.

As especificações FT-CORBA atendem apenas a aspectos básicos de tolerância a falhas, definindo algumas interfaces genéricas, de fácil entendimento, e úteis a aplicações que necessitam de tolerância a falhas em sistemas distribuídos. Apenas alguns compromissos foram assumidos pela OMG. Em particular, para prover interoperabilidade entre produtos de fabricantes diferentes, outras interfaces e protocolos devem ser definidos posteriormente. Por enquanto, segundo a OMG, as necessidades que não são supridas por esta especificação podem ser atendidas por soluções proprietárias.

A forma de redundância, pela qual o padrão FT-CORBA provê tolerância a falhas, é a replicação de objetos. Através de replicação de componentes importantes é possível ter maior confiabilidade no sistema e aumentar a disponibilidade dos recursos. O FT-CORBA provê suporte à redundância permitindo que clientes façam requisições a diferentes réplicas de um objeto servidor utilizando a mesma chamada. Suporta um conjunto de estratégias de tolerância a falhas, incluindo técnicas de replicação e definição de propriedades de tolerância a falhas para cada grupo de objeto replicado.

Nos itens a seguir veremos uma breve descrição do padrão FT-CORBA, conforme especificação descrita em [OMG00], e a relação que podemos fazer dos componentes definidos neste padrão com o serviço de diagnóstico.

Os conceitos básicos e a arquitetura do padrão serão descritas resumidamente no item 6.1 desta dissertação. No item 6.2 apresentaremos as áreas de gerenciamento para as quais foram especificados protocolos e interfaces, a saber: gerenciamento de replicação,

gerenciamento de falhas e gerenciamento de recuperação e *logging*. Daremos uma ênfase especial à área de gerenciamento de falhas por ser esta parte da especificação que mais nos interessa. No item 6.3 faremos um paralelo entre componentes especificados no padrão FT-CORBA e os propostos na arquitetura do SDF. Mostraremos como a infraestrutura do SDF foi adaptada para monitorar objetos que implementam a interface *PullMonitorable* do padrão FT-CORBA. Finalizaremos, no item 6.4, fazendo um resumo dos principais tópicos vistos neste capítulo.

6.1. Infraestrutura do Padrão FT-CORBA

No padrão FT-CORBA, para tornar um objeto tolerante a falhas, várias réplicas são criadas e gerenciadas como um grupo de objetos. Cada objeto individual tem sua referência e o grupo todo tem uma referência de grupo, IOGR (*Interoperable Object Group Reference*). Esta referência ao grupo é usada no momento em que um cliente necessita solicitar um serviço a um objeto replicado. Independente do tipo de replicação adotada (passiva, ativa, semi-ativa) por causa da abstração do grupo de objetos, os clientes não ficam cientes que existem réplicas do objeto e que acontecem falhas e recuperação em algumas destas réplicas. Além disto, para cada grupo de objetos é associado um conjunto de propriedades de tolerância a falhas. É possível definir propriedades que se aplicam a todos os objetos do grupo ou então a todos os objetos de um tipo específico.

Muitas aplicações que necessitam de tolerância a falhas são grandes e complexas, sendo inapropriado manipular este tipo de aplicação como uma única entidade. O conceito de domínios de tolerância a falhas foi introduzido para facilitar o gerenciamento deste tipo de aplicação. Cada domínio de tolerância a falhas consiste em várias máquinas e grupos de objetos. Associado a cada domínio está um conjunto dos componentes definidos na infraestrutura de tolerância a falhas FT-CORBA (*replication manager, fault notifier e fault detectors*), os quais atuam exclusivamente dentro deste domínio. Segundo a especificação, um objeto pertence apenas a um grupo e domínio de tolerância a falhas. Grupos de objetos têm objetos em diversas máquinas.

A figura 6.1 mostra dois diferentes domínios de tolerância a falhas, DOM1 e DOM2, onde existem grupos de objetos replicados. Cada domínio de tolerância a falhas é representado por um círculo tracejado na figura. O grupo A é composto pelos objetos A1, A2 e A3, o grupo B pelos objetos B1, B2 e B3, o grupo C pelos objetos C1 e C2 e o grupo de objetos D pelos objetos D1 e D2. O domínio DOM1 contém os grupos A e C e o domínio DOM2 contém os grupos B e D. As máquinas 1 e 2 estão no domínio DOM1, as máquinas 4, 5 e 6 estão no domínio DOM2 e a máquina 3 pertence aos dois domínios já que o objeto A2 está no domínio DOM1 e o objeto B2 faz parte do domínio DOM2.

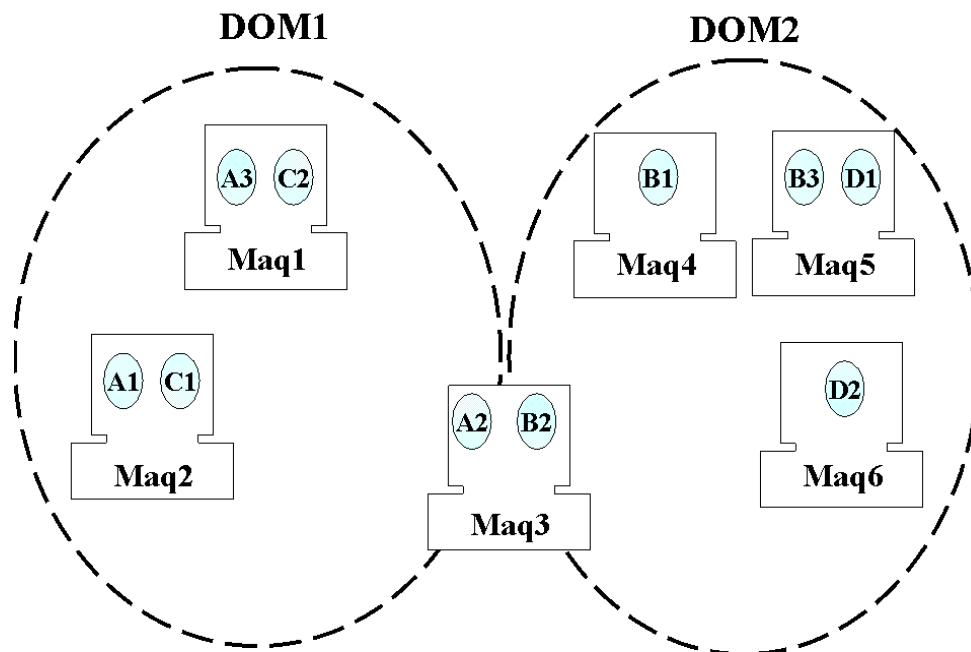


Figura 6.1 - Domínios de tolerância a falhas com seus grupos de objetos replicados.

Para cada domínio de tolerância a falhas existe um *replication manager*, um *fault notifier* e *fault detectors*. A nível lógico, existe apenas uma instância do *replication manager* e do *fault notifier* para cada domínio de tolerância a falhas. Mas, fisicamente, eles estão replicados para proteger contra falhas, da mesma forma que os objetos da aplicação. *Fault*

detectors estão divididos em *object fault detectors*, *process fault detectors* e *host fault detectors*.

Todos os tipos de *fault detector* são baseado em *timeout*. Os *object fault detectors* são monitorados por *process fault detectors* e estes, por sua vez, são monitorados pelo *host fault detector*, que é replicado para tolerar falhas. Todos os tipos de *fault detectors* reportam falhas para o *fault notifier*, que repassa estas notificações para o *replication manager*, *fault analyzer* e para outros *consumers* que estejam registrados. Por eficiência, os *fault detectors* que monitoram os objetos da aplicação são tipicamente localizados na mesma máquina que o objeto.

A especificação FT-CORBA para gerenciamento de falhas não limita o número nem a forma como estão arrumados estes *fault detectors* em um domínio de tolerância a falhas. Diferentes tipos de arranjos são possíveis. Em um sistema grande, com muitas máquinas, eles podem estar arrumados em uma estrutura hierárquica por uma questão de escalabilidade.

Levando em consideração o domínio DOM1 mostrado na figura 6.1, apresentamos os componentes da arquitetura FT-CORBA neste domínio de tolerância a falhas na figura 6.2. Por motivo de simplificação representamos apenas as máquina 1 e 2 do domínio DOM1. Nesta figura vemos os três tipos de *fault detectors* (*object*, *process*, *host*) e o *fault notifier*. Vemos também o *fault analyzer*, o *replication manager* e os *consumers*, que se registram no notificador para receber informações de eventos de falhas. Estes componentes serão explicados detalhadamente nos itens 6.2.1 e 6.2.2. As setas tracejadas da figura 6.2 indicam monitoria e as setas contínuas indicam eventos de falhas que estão sendo reportados.

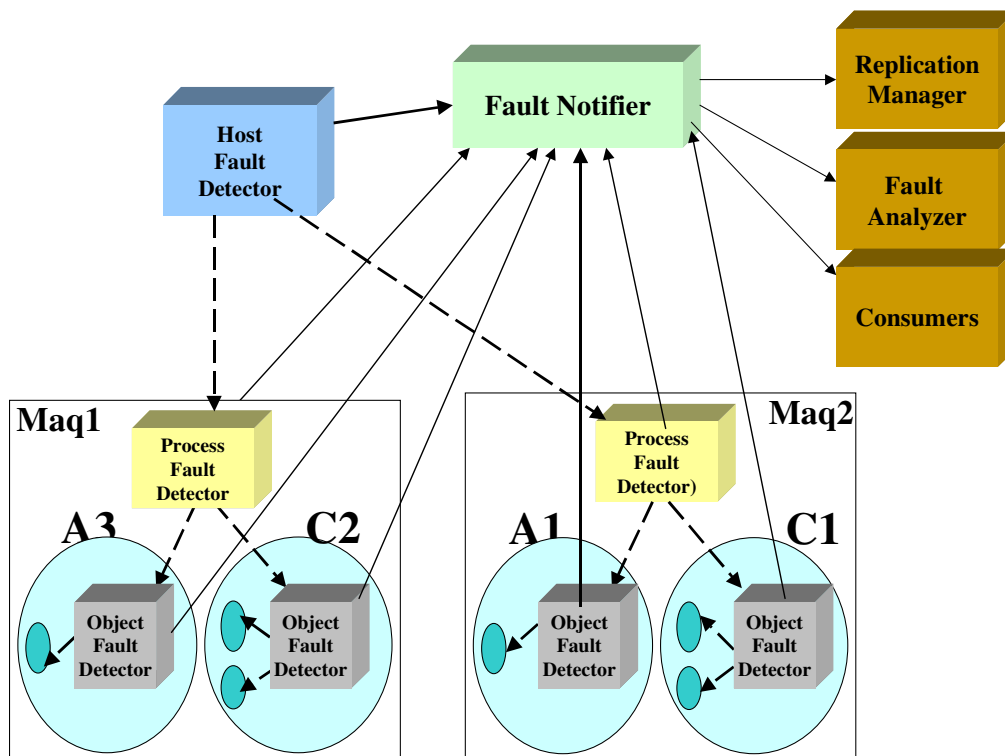


Figura 6.2 - Componentes definidos na arquitetura de tolerância a falhas FT-CORBA

No item seguinte veremos as áreas onde foram especificados os componentes da infraestrutura FT-CORBA.

6.2. Serviços de Gerenciamento da Especificação FT-CORBA

A infraestrutura proposta na especificação de Tolerância a Falhas CORBA é composta de vários componentes, como visto no item anterior, que são implementações de objetos CORBA. Estes componentes dividem-se nas áreas onde o padrão FT-CORBA especificou protocolos e interfaces, a saber: gerenciamento de replicação, gerenciamento de falhas e gerenciamento de recuperação e *logging*. O gerenciamento de replicação é composto pelos serviços de gerenciamento de propriedades, gerenciamento de grupo de objetos e fábrica genérica. No gerenciamento de falhas temos a definição das interfaces de detecção de falhas, de notificação de falhas e de análise de falhas. E por fim, o gerenciamento de

recuperação e *logging* define os mecanismos para transferência de estado de objetos e recuperação de réplicas que falharam.

A seguir, faremos um resumo dos principais componentes e interfaces dos tipos de gerenciamento especificado no padrão FT-CORBA.

6.2.1. Gerenciamento de Replicação

No gerenciamento de replicação, a especificação FT-CORBA define o componente *replication manager*. É responsabilidade do *replication manager* a manutenção dos objetos replicados existentes no domínio de tolerância a falhas ao qual ele está vinculado. Ele controla a criação de novas réplicas (entradas) e as saídas, normais ou por falha, de réplicas de um grupo de objetos. Na criação de novas réplicas o *replication manager* utiliza o objeto fábrica genérica, que negocia com os objetos fábricas locais a criação de uma nova réplica nas diferentes máquinas.

Cada grupo de objetos tem associado um conjunto de propriedades, no domínio de tolerância a falhas, que podem ser definidas quando o grupo é criado, ou mesmo enquanto a aplicação executa seu processamento. Estas propriedades incluem: o tipo de replicação existente no grupo, *ReplicationStyle*; o tipo de adição de um objeto ao grupo, *MembershipStyle*; se é responsabilidade da aplicação ou da infraestrutura de tolerância a falhas manter a consistência do grupo, *ConsistencyStyle*; o estilo da monitoria sobre o objeto (interrogando-o ou então recebendo indicações periodicamente de que este está ativo), *FaultMonitoringStyle*; a forma da monitoria sobre o grupo (sobre todos os membros do grupo ou sobre objetos representativos), *FaultMonitoringGranularity*; o número de objetos a serem criados inicialmente e o número mínimo de réplicas de objetos do grupo, *InitialNumberReplicas* e *MinimumNumberReplicas*.

Além destas propriedades de tolerância a falhas, no gerenciamento de replicação existe ainda especificado *FaultMonitoringIntervalAndTimeout*, que indica o intervalo de tempo entre os *pings* sobre um objeto e *CheckpointInterval*, que indica o intervalo de tempo para um cada atualização de estados.

O *replication manager* deve implementar três interfaces definidas na especificação FT-CORBA: *PropertyManager*, *ObjectGroupManager* e *GenericFactory*. Os métodos herdados da interface *PropertyManager* permitem a definição de propriedades associadas com grupos de objetos criados pelo *replication manager*. A interface *ObjectGroupManager* define métodos que permitem à aplicação exercer controle sobre a adição, remoção e localização dos membros do grupo de objetos. Os métodos herdados de *GenericFactory* permitem ao *replication manager* criar e excluir membros de grupos de objetos.

6.2.2. Gerenciamento de Falhas

Na especificação FT-CORBA, o gerenciamento de falhas engloba as atividades de detecção de falhas (detectar a presença de falhas e gerar relatórios sobre estas falhas), notificação de falhas (propagar os relatórios de falhas às entidades registradas para recebê-los), e análise de falhas (analisar os relatórios de falhas gerados resumindo-os em relatórios condensados).

Segundo o padrão, um sistema tolerante a falhas tem vários detectores que, geralmente, são baseados em *timeouts*. Existem duas formas de monitoria: *pull* e *push*. Na monitoria *pull*, o *fault detector* interroga periodicamente o objeto monitorado para determinar se este está vivo. Na monitoria *push*, o objeto monitorado periodicamente envia uma mensagem de que está vivo ao *fault detector*. Este último tipo de monitoria não foi tratado nesta primeira especificação FT-CORBA.

A monitoria do tipo *pull* foi especificada no padrão através da interface *PullMonitorable* que deve ser implementada pelos objetos a serem monitorados. O método *is_alive()*, definido nesta interface, é invocado pelo *fault detector*, dentro da infraestrutura de tolerância a falhas, para determinar se um objeto do grupo está ativo ou se falhou. As falhas tratadas são do tipo *crash*.

Na infraestrutura de tolerância a falhas, os *fault detectors* detectam falhas em objetos e reportam estas falhas ao *fault notifier*. O *fault notifier* recebe os relatórios, filtra as informações a fim de eliminar relatórios duplicados, propagando-os como notificações de eventos de falhas aos *consumers* (inscritos para receberem estas notificações), ao *fault*

analyzer e ao *replication manager*. O *fault analyzer* analisa os relatórios recebidos produzindo relatórios resumidos. Este componente é especialmente importante em aplicações complexas por sua capacidade de correlacionar eventos ao analisar os relatórios que foram gerados. Quando o *fault analyzer* faz a correlação entre os relatórios ele executa esta atividade de acordo com a aplicação em questão. Como estas atividades são específicas da aplicação ou do ambiente, o desenvolvedor da aplicação é responsável pelo algoritmo empregado pelo *fault analyzer*. Os relatórios resumidos produzidos pelo *fault analyzer* são enviados ao *fault notifier*. Este, por sua vez, dissemina a informação para os *consumers* registrados.

A interface *FaultNotifier*, definida na especificação FT-CORBA, deve ser implementada pelo componente *fault notifier*. Esta interface contém: métodos que permitem aos *fault detectors* e ao *fault analyzer* enviar relatórios de falhas ao *fault notifier*, métodos que possibilitam ao *replication manager* e a outros *consumers* se inscreverem para serem notificados quando ocorrerem eventos de falhas e métodos para disseminar as informações de falhas aos *consumers*. Segundo o padrão, a própria aplicação também pode se inscrever para receber notificações de eventos de falhas.

6.2.3. Gerenciamento de Recuperação e *Logging*

A infraestrutura de tolerância a falhas especificada no padrão FT-CORBA inclui mecanismos para gerenciamento de recuperação e *logging* de réplicas em um grupo. Durante uma operação normal de um grupo, com esquema de replicação passiva, o mecanismo de *logging* guarda o estado e as ações do objeto primário do grupo. Se este objeto falhar, o mecanismo de recuperação lê os registros do log e utiliza-os para restaurar o estado do objeto primário naquele que fará o papel de backup, tornando-o apto para continuar a prover o serviço. Os mecanismos de *logging* e recuperação também são usados para ativar um novo membro de um grupo com replicação ativa. Não existem interfaces definidas para os mecanismos de *logging* e recuperação pois estes mecanismos nunca são invocados diretamente por programas da aplicação. Existem, no entanto, interfaces que

objetos da aplicação precisam implementar para possibilitar logging e recuperação de estado. Estas interfaces são *Checkpointable* e *Updateable*.

6.3. Componentes FT-CORBA e o SDF

O escopo do serviço de diagnóstico vai da detecção até o diagnóstico da falha, com a reconfiguração do grupo, contudo não atinge aspectos de replicação de componentes. O SDF trabalha com processos que não são objetos de um grupo de réplicas, mas são implementações independentes de objetos/serviços necessários para o funcionamento de uma aplicação distribuída. Assim, o ponto em comum entre o SDF e os serviços especificados no padrão FT-CORBA é a detecção de falhas de componentes de um grupo. Os objetivos por detrás destas detecções, no entanto, são diferentes. O objetivo dos serviços FT-CORBA é gerenciar um grupo de réplicas, recuperando-as quando há falhas detectadas. O SDF por sua vez, tem por objetivo manter a aplicação informada sobre o estado dos processos do grupo (correto, suspeito ou falho) e sobre o tempo atual de comunicação entre o módulo SDF e os processos. Isto auxilia a aplicação a determinar quais processos/serviços devem ser utilizados no momento.

A figura 6.3 apresenta um paralelo entre os componentes especificados no padrão FT-CORBA e os componentes do SDF. Na parte superior da figura vemos os componentes especificados no padrão FT-CORBA, enquanto na parte inferior podemos observar os componentes do SDF. O *fault detector* na especificação FT-CORBA corresponde, na arquitetura do SDF, aos objetos agente de conexão e agente de objetos (número 1 da figura). Processos e objetos que implementam a interface de gerenciamento do SDF, *ObjetosSistemas*, ou então que implementam a interface de monitoria do padrão FT-CORBA, *PullMonitorable*, podem ser investigados pelo agente de conexão e o agente de objetos do SDF, conforme pode ser visto na figura 6.3. O papel descrito para o *fault notifier* é executado, no SDF, pelo gerente (número 2 da figura) na arquitetura SDF. Este gerente encaminha as falhas detectadas para o objeto reconfiguração enquanto que, no padrão FT-CORBA, o *fault notifier* repassa as falhas detectadas para o *replication manager*, o *fault analyzer* (em aplicações complexas) e outros *consumers* registrados.

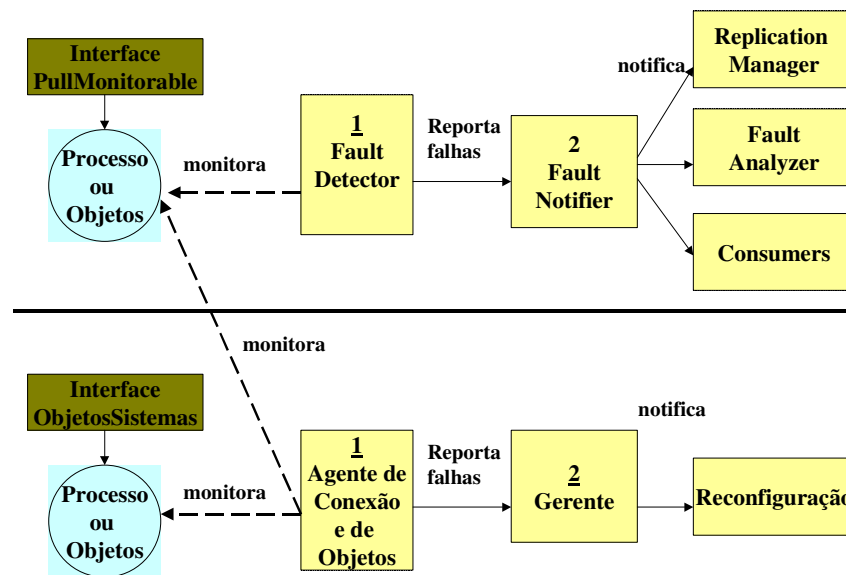


Figura 6.3 - Componentes FT-CORBA comparados aos componentes do SDF

A seguir mostraremos como adaptamos o SDF para que este esteja de acordo com o padrão FT-CORBA, no que se refere às atividades de detecção de falhas. Apresentamos a classe *ProcessoInterfaceCorba* que foi desenvolvida para permitir ao SDF monitorar processos que implementam a interface *PullMonitorable* do FT-CORBA.

6.3.1. Classe *ProcessoInterfaceFTCORBA*

Como visto na figura 6.3, o SDF pode monitorar objetos FT-CORBA para detectar quando ocorrerem falhas. Essa monitoria dos objetos que implementam a interface *PullMonitorable* acontece através de um processo interface (a definição pode ser vista no capítulo 3).

A classe *ProcessoInterfaceFTCORBA* foi desenvolvida para atuar como processo interface. Esta classe tem um atributo que é a referência ao objeto FT-CORBA a ser monitorado. Seu método *respondeMonitoria()* aciona internamente o método *is_alive()* do objeto FT-CORBA existente como atributo.

Na figura 6.4 podemos ver o SDF monitorando três processos de uma aplicação distribuída. O processo P2 funciona como um processo intermediário que permite ao SDF monitorar o objeto FT-CORBA. Este objeto implementa a interface de monitoria *PullMonitorable*, como pode ser visto na figura.

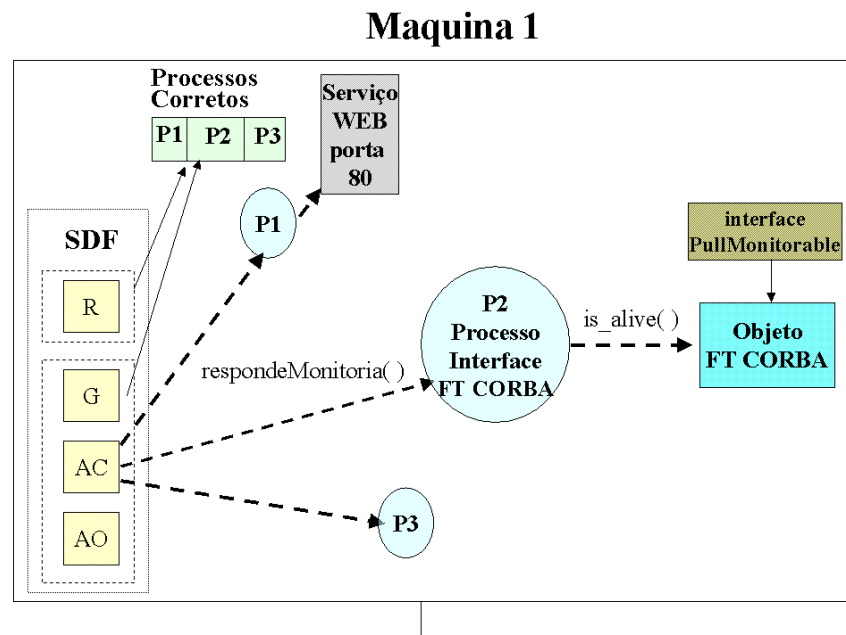


Figura 6.4 - Monitoria de processo que implementa a interface FT-CORBA *PullMonitorable*

Na implementação atual do SDF, para que um objeto FT CORBA seja monitorado é necessário que este esteja inscrito no servidor de objetos, assim que for instanciado. O *ProcessoInterfaceFTCORBA* é informado pelo usuário sobre o valor do PID do objeto FT-CORBA associado a ele. Com esta informação, este aciona o servidor de objetos para pegar a referência ao objeto FT-CORBA colocando-a no seu atributo. A partir deste momento a monitoria pode ser inicializada pelo SDF.

A classe *ProcessoInterfaceFTCORBA*, como todas as outras classes que funcionam como processos interfaces, implementa a interface *ObjetosSistemas* do SDF. A figura 6.5 mostra a classe *ProcessoInterfaceFTCORBA* junto com outras classes desenvolvidas neste trabalho. Estas classes têm por objetivo servir de ponte para processos/serviços que devem ser monitorados mas que não implementam a interface de monitoria *ObjetosSistemas*.

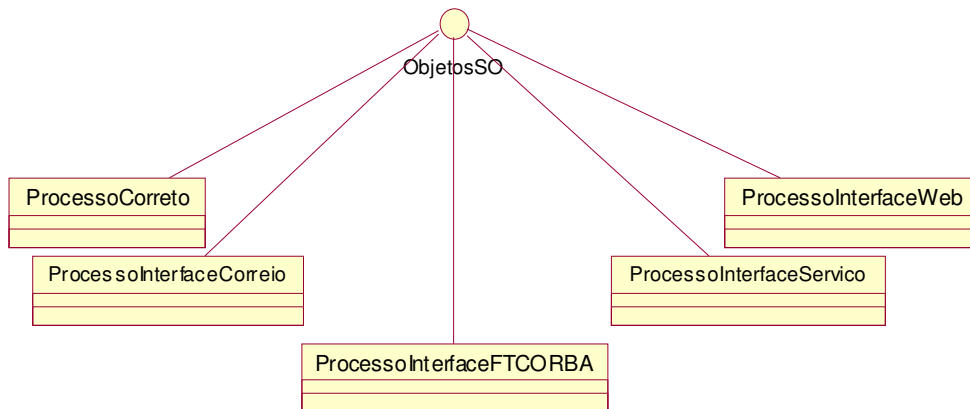


Figura 6.5 - Classes de processos interfaces utilizadas no SDF

6.4. Resumo dos Principais Itens Vistos Neste Capítulo

O padrão FT-CORBA foi especificado para atender aspectos de tolerância a falhas, suportando o desenvolvimento de aplicações que necessitam de um alto grau de confiabilidade.

Além de definir algumas interfaces genéricas úteis para este tipo de aplicação, a especificação define o conjunto dos componentes da infraestrutura de tolerância a falha que são: o *replication manager*, o *fault notifier* e os *fault detectors (object, process, host)*. Estes componentes estão associados a domínios de tolerância a falhas onde, em cada domínio, existem máquinas e grupos de objetos.

Neste capítulo descrevemos resumidamente os serviços de gerenciamento definidos no padrão, onde foram especificados estes componentes da infraestrutura. Concentramo-nos no gerenciamento de falhas por ter mais relação com o nosso trabalho. Apresentamos um paralelo entre os componentes FT-CORBA e a arquitetura SDF. Os *fault detectors*, propostos no padrão FT-CORBA, são baseados exclusivamente em *timeout*, sendo portanto imprecisos para ambientes com características assíncronas.

Mostramos como adaptamos o serviço de diagnóstico para monitorar objetos que implementam, a interface *PullMonitorable*. O objetivo foi tornar o SDF de acordo com o

padrão FT-CORBA no que se refere à detecção de falhas. Apresentamos a classe `ProcessoInterfaceFTCORBA` que permite a monitoração de objetos FT-CORBA pelo SDF.

No próximo capítulo analisaremos trabalhos correlatos que se propõem a executar detecção de falhas e diagnóstico sobre plataforma CORBA. Alguns destes trabalhos fazem referência ao padrão FT-CORBA.

Capítulo 7

Trabalhos Correlatos

Neste capítulo apresentaremos alguns trabalhos correlatos, comparando-os com o serviço proposto nesta dissertação.

7. Trabalhos Correlatos

Muitos trabalhos tratam de detecção de falhas de componentes em sistemas distribuídos. Contudo, delimitamos nossa análise de trabalhos correlatos aos que se propõem a realizar detecção e diagnóstico de falhas sobre plataforma CORBA. Neste capítulo analisaremos quatro trabalhos comparando com o Serviço de Diagnóstico de Falhas.

Os trabalhos apresentados nos itens 7.1, 7.2, 7.3, respectivamente, *GroupPac* [LF00, LFFO00, LFPS01], *DOORS* [CHY98, NGYS00] e *OGS* [Fel98], baseiam-se na concepção de serviços CORBA. Eles se utilizam da infraestrutura fornecida pelo ORB comum especificado pela OMG [OMG96]. O trabalho apresentado em 7.4, *Piranha* [Maf97], apresenta uma ferramenta para gerenciamento de aplicações CORBA que funciona sobre um ORB modificado, o *Electra* [Maf95]. A abordagem, neste caso, foi a de integração [FGG97], onde uma biblioteca de comunicação em grupo é adicionada às funcionalidades do ORB padrão. Finalizaremos o capítulo apresentando, no item 7.5, um resumo comparativo entre o SDF e os trabalhos correlatos descritos nos itens anteriores.

7.1. GroupPac

O GroupPac [LF00, LFFO00, LFPS01] consiste em um conjunto de serviços, para o desenvolvimento de aplicações tolerante a falhas. Estes serviços têm interfaces baseadas nas especificações FT CORBA descritas no capítulo anterior. Apresentam um conjunto de extensões e adaptações com a finalidade de atender requisitos de tolerância a falhas em sistemas assíncronos e de larga escala [LFPS01]. Algumas extensões propostas foram incluídas no serviço de gerenciamento de falhas. Um protocolo de consenso para detecção de falhas de objetos foi especificado visando ambientes assíncronos. Propõe um conjunto de detectores, dispostos na rede, onde todos os detectores monitoram todos as máquinas dentro do domínio de tolerância a falhas definido.

O serviço de detecção do GroupPac atua sobre dois níveis de monitoramento: detectores e máquinas. No primeiro nível, os detectores formam um anel virtual onde, periodicamente, cada membro do anel monitora o parceiro imediatamente anterior na seqüência do anel. Desta forma são controladas as entradas e saídas dos detectores membros do grupo. Quando um detector suspeita de outro, o protocolo de *membership* é ativado e o detector primário difunde uma mensagem para detectar quem ainda continua no grupo. Todos os detectores ativos devem dar um reconhecimento a esta mensagem (*Ack*). Após um prazo de espera pré-definido, se necessário, o detector primário produz uma nova lista dos membros do grupo de detectores, a partir dos *Acks* recebidos. Quando a falha é no detector primário, o candidato a substituí-lo segue a ordem implementada pelo anel virtual.

O outro nível de detecção de falhas do GroupPac é a nível de máquina, onde os detectores de defeitos¹³ atuam sobre todas as máquinas do domínio. Para isso, os detectores monitoram periodicamente, dentro de um intervalo de tempo, as mesmas máquinas. Se um deles suspeitar de uma máquina, avisa ao detector primário, que em seguida, solicitará a todos os detectores do grupo que informe o estado desta máquina no próximo intervalo de monitoração. O detector primário, com estas informações, decide sobre o estado da máquina suspeita. O protocolo é baseado em voto majoritário para alcançar o consenso. Na falha de uma máquina, são considerados falhos todos os seus processos e objetos. Quando existe consenso sobre a falha de uma máquina, o detector primário avisa ao notificador de falhas. Este, por sua vez, informa ao gerenciador de replicação para que haja a remoção da máquina do grupo e uma nova referência do grupo, IOGR, seja gerada. A nova IOGR é enviada aos detectores para que estes atualizem suas listas de máquinas a serem monitoradas.

A figura 7.1 representa os dois níveis de monitoria executados pelos serviços GroupPac. O primeiro nível é a monitoria dos próprios detectores de defeitos (DD) do grupo (representada na figura pelas setas contínuas). Nesta figura vemos o detector de defeitos da máquina M1 monitorando o detector de M3, o detector de M3 monitorando o detector de M2 e, para finalizar o anel, o detector de defeitos da máquina M2 monitorando o detector de M1. Qualquer quebra deste anel é percebida e uma nova lista de detectores de defeitos é

¹³ No texto original, os autores utilizam a notação “detectores de falhas”. Para se tornar coerente com a terminologia adotada nesta dissertação utilizamos “detectores de defeitos”.

gerada. O outro nível de monitoria mostrado nesta figura é sobre as máquinas. Este tipo de monitoria é representado na figura 7.1 através das setas tracejadas. Cada detector de defeitos monitora todas as máquinas que estão no seu domínio. Assim, o detector de defeitos da máquina M1 monitora as máquinas M2, M3 e M4, o detector de M2 monitora as máquinas M1, M3 e M4, e o detector de M3 monitora as máquinas M1, M2 e M4. Se algum dos detectores suspeitar de uma máquina este avisa ao detector primário (DD Pri) para iniciar o protocolo de acordo a fim de retirar a máquina do grupo. Como pode ser visto na figura pelas setas pontilhadas, o notificador de falhas (NF) recebe a informação de falha de uma máquina do detector primário (DD Pri), repassando a informação para o gerente de replicação (GR). Com esta informação uma nova IOGR é gerada, como mostra a figura.

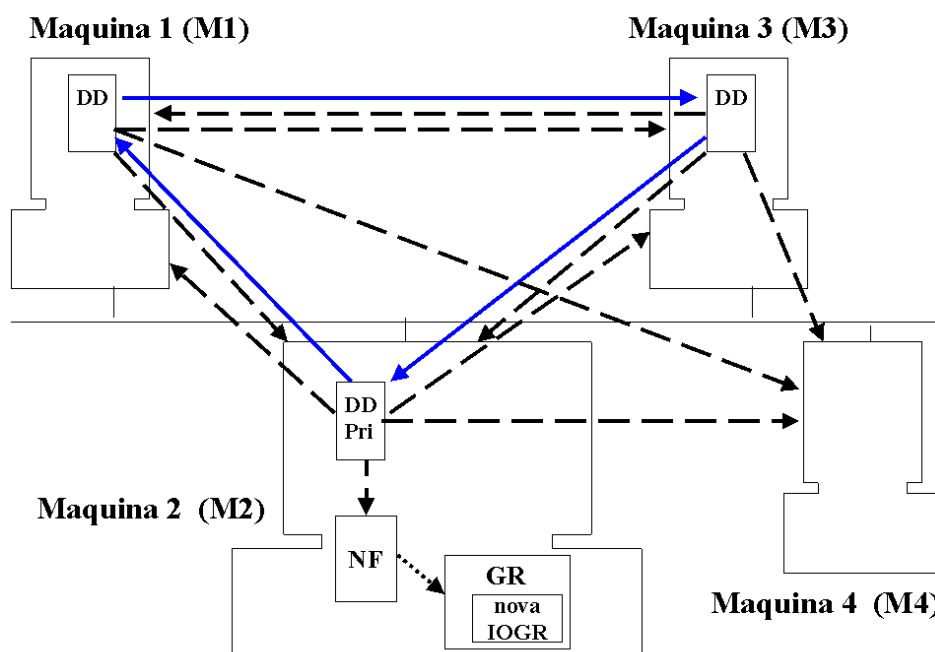


Figura 7.1 - Arquitetura dos serviços GroupPac

O GroupPac propõe soluções com o objetivo de facilitar a escalabilidade [LFPS01]. Objetos de diferentes domínios de tolerância a falhas podem ser localizados e se juntar ou sair de um grupo dinamicamente. Isso é feito baseado na associação de domínios de tolerância a falhas com uma estrutura hierárquica de servidores de nomes.

O serviço de nomes no GroupPac é baseado em uma hierarquia formada de um serviço de nomes global, replicado por questões de tolerância a falhas, e serviços de nomes locais em cada domínio de tolerância a falhas. Réplicas do serviço de nomes global podem ser instaladas em máquinas de domínios locais por uma questão de eficiência. Foram especificadas algumas propriedades adicionais à especificação FT CORBA que estendem as propriedades de tolerância a falhas para dar suporte a grupos de diferentes domínios.

7.1.1. Comparação dos Serviços GroupPac com o SDF

O SDF não é um serviço para detectar falhas de réplicas de objetos de uma aplicação e prover sua recuperação, como os serviços do GroupPac. Nossos objetivos no SDF vão da detecção até o diagnóstico da falha, não atingindo aspectos de replicação de componentes. No entanto, as atividades executadas pelo SDF se assemelham às atividades de detecção especificadas nos serviços GroupPac.

O serviço de detecção de falhas especificado pelo GroupPac monitora apenas dois níveis: detectores de defeitos e máquinas. Os processos existentes nas máquinas do grupo são considerados falhos se a máquina onde estes residem falhar. No entanto, não existe, nos serviços do GroupPac, uma preocupação em detectar o defeito de processos individuais em uma máquina. Na proposta FT CORBA esta necessidade foi atendida com a especificação dos detectores de objetos/processos. Neste padrão, este tipo de detector reside na máquina onde estão os processos e são monitorados por detectores de máquinas. Detectores de processos são importantes pois, muitas vezes, um processo pára a sua execução sem que a máquina tenha tido algum problema. Em um ambiente distribuído, com objetos espalhados em máquinas distintas na rede a existência de detectores de defeitos de processos é uma necessidade. No SDF, além da detecção de falhas de máquina (item 3.5.3) e módulos do serviço (item 3.5.4), existe também a detecção de falhas individuais de processos (item 3.5.2). Desta maneira é possível ter, de uma forma mais precisa, a informação dos processos que falharam e o motivo da falha.

O GroupPac melhora a proposta FT CORBA para detecção de falhas em ambientes assíncronos na medida em que especifica um protocolo de acordo para retirada de máquinas do grupo, ao invés de considerar uma máquina falha se esta não responder a pelo menos um detector dentro de um *timeout*. Assim como o GroupPac, também propomos um protocolo de acordo, tolerante a falha do coordenador, para retirada de processos que residem em máquinas falhas. O funcionamento deste protocolo de acordo foi descrito no item 3.6 dessa dissertação.

7.2. DOORS

O DOORS (*Distributed Object-Oriented Reliable Service*) [CHY98, NGYS00] consiste em um *framework* desenvolvido nos laboratórios da Bell que foi implementado como um serviço CORBA para prover tolerância a falhas. Este *framework* atinge os objetivos propostos através da detecção de falhas de máquinas e de objetos. Muitos dos conceitos apresentados neste *framework* foram incorporados na especificação do padrão FT CORBA, uma vez que seus criadores contribuíram na definição deste padrão. A seguir descreveremos o *framework* DOORS original e depois as adaptações para torna-lo uma implementação de acordo com a especificação FT CORBA.

No *framework* DOORS original [CHY98] foram especificados três módulos que juntos provêm detecção de falhas e recuperação de objetos replicados de uma aplicação.

O primeiro módulo é chamado de *ReplicaManager*. Ele é responsável pelo gerenciamento de réplicas de objetos, controlando a inicialização e a migração de réplicas durante falhas. Para cada objeto da aplicação existe uma tabela no *ReplicaManager* com informações das máquinas onde este objeto pode ser ativado, o estilo de replicação, o estado de cada replica existente, entre outras informações. O *ReplicaManager* é centralizado e existe apenas um objeto deste tipo rodando na rede. Por uma questão de robustez é proposto um esquema de *checkpointed* para este módulo. Todos os objetos da aplicação se registram no *ReplicaManager* inicialmente.

O segundo módulo é chamado de *WatchDog* e roda em cada máquina da rede detectando falhas de objetos residentes na máquina. Este módulo suporta dois mecanismos de detecção de falhas, a saber: *pulling* e *heartbeat*. Em detecções do tipo *pulling*, o *WatchDog* envia periodicamente mensagens do tipo *ping* aos objetos da aplicação. Se o *ping* falhar, ele assume que o objeto falhou. Já em detecções com *heartbeat*, se o *WatchDog* não receber um *heartbeat* dentro de um certo intervalo de tempo, o objeto é considerado falho. Quando o *WatchDog* detecta uma falha de um objeto da aplicação ele reporta esta falha ao *ReplicaManager*.

O terceiro módulo é o *SuperWatchDog* que também é centralizado e é responsável pela detecção de falhas de máquinas. Todos os objetos *WatchDog* existentes na rede se registram no objeto *SuperWatchDog* para possibilitar a detecção de falhas de máquinas. A estratégia de detecção utilizada é o *heartbeat*. Todos os objetos *WatchDog* enviam periodicamente heartbeats para o *SuperWatchDog*. Quando isso não acontece, então, o *SuperWatchDog* assume que a máquina falhou. Ele avisa ao *ReplicaManager* a falha e inicia a recuperação dos objetos da máquina falha em outras máquinas da rede. O *SuperWatchDog* é também responsável por notificar as aplicações interessadas quando ocorrem falha de máquinas. Para evitar um ponto único de falha, o *SuperWatchDog* é replicado formando uma estrutura em anel lógico, onde cada um envia periodicamente *heartbeats* para seu vizinho no anel. Nesta versão não foi especificado um protocolo de acordo para eleição de um novo objeto *SuperWatchDog* primário quando há uma falha. O *SuperWatchDog* que deveria receber o *heartbeat* do objeto primário é quem assume este papel. O *ReplicaManager* é localizado na mesma máquina do objeto *SuperWatchDog* primário. Se há uma falha nesta máquina então um objeto *ReplicaManager* é iniciado na máquina que contém o novo *SuperWatchDog* primário do grupo.

Como o *framework* DOORS foi desenvolvido antes do padrão FT CORBA, uma nova versão foi proposta em [NGYS00]. Esta versão implementa um subconjunto das funcionalidades da especificação FT CORBA. O objeto *WatchDog* chama-se, na nova versão do DOORS, de *FaultDetector* e o *SuperWatchDog* é representado pelo *Super FaultDetector*. As funções destes componentes permaneceram as mesmas, de forma que o primeiro detecta falhas em processos e o segundo detecta falhas de máquinas. Não existe o componente FT CORBA *fault notifier*. Tanto o *FaultDetector* como o *Super FaultDetector*

atuam como notificadores propagando os relatórios de falhas ao gerente de replicação, chamado de *ReplicationManager*. Os *FaultDetectors* suportam monitoria baseada em *pull* e *push*. Tanto ele como o *ReplicationManager* são monitorados pelo *Super FaultDetector* através de *heartbeats*. Como a especificação FT CORBA não permite ponto único de falhas, os *Super FaultDetector* são replicados e enviam periodicamente *heartbeats* entre si. Se o *Super FaultDetector* primário falhar, os outros objetos backups elegem o novo detector primário.

Na figura 7.2 vemos as máquinas M1, M2 e M3 com os objetos replicados O1 e O2. Em cada uma das máquinas existe um *FaultDetector* (FD) para monitorar os objetos replicados. Esta monitoria é representada na figura pelas setas tracejadas. Estes detectores enviam mensagens do tipo *heartbeat* para o *Super FaultDetector* (Super FD) que está localizado na máquina M2, conforme pode ser visto na figura pelas setas pontilhadas. Em M2 existe também o *ReplicationManager* (RM), que mantém o gerenciamento das réplicas dos objetos com a lista das máquinas onde cada réplica está localizada. Este gerente também envia mensagens de *heartbeat* para o *Super FaultDetector* e recebe as notificações de falhas que ocorreram de todos os detectores. Na figura 7.2 estas notificações são representadas pelas setas contínuas.

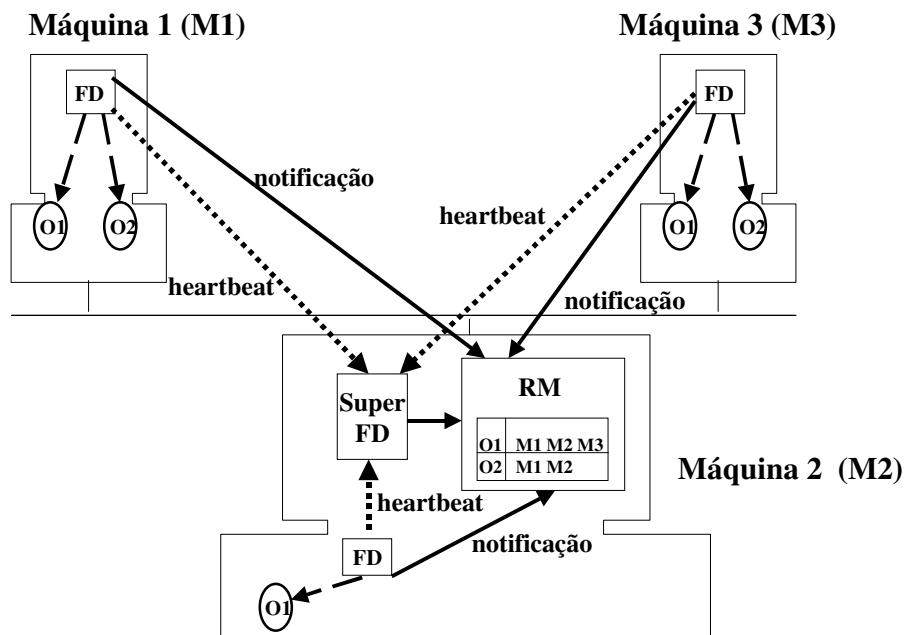


Figura 7.2 - Arquitetura DOORS

7.2.1. Comparação dos Serviços DOORS com o SDF

As atividades de detecção do DOORS, como no serviço de diagnóstico, acontecem em três níveis: máquina, objetos/processos individuais e detectores de defeitos. No entanto, no DOORS, a monitoria de objetos/processos pode acontecer por *pulling* ou então por *heartbeat* enquanto que no SDF isso acontece através de *pulling*.

No SDF todos os processos existentes nas máquinas do grupo são monitorados por todos os detectores de defeitos. O objetivo é obter o tempo atual de comunicação com cada processo, seja ele local ou remoto, ao passo que falhas locais ou remotas são detectadas. No DOORS, os *FaultDetector* de cada máquina monitoram apenas os processos existentes na máquina onde residem. Assim, cada *FaultDetector* detecta apenas falhas de processos locais.

A forma de detectar falhas de máquinas também é executada de forma diferente, quando comparamos o SDF com o DOORS. No DOORS, os *FaultDetector* nas máquinas enviam periodicamente mensagens de *heartbeats* para indicar *Super FaultDetector* que a máquina está ativa. No SDF esta monitoria é feita pelos mesmos detectores de defeitos de processos através de solicitações ao serviço ECHO da máquina. É a resposta deste serviço que determina se a máquina está ativa ou não.

No DOORS, se o detector de processos falhar, a máquina e os processos residentes nela serão considerados falhos, sem que isto tenha necessariamente ocorrido. No SDF, quando há uma falha no módulo detector, os processos desta máquina continuam a ser monitorados sem interrupção pelos outros módulos SDF até que um dos processos se torne falho.

O SDF e o DOORS têm objetivos distintos, pois o DOORS se propõe a prover uma infraestrutura de suporte para tolerar falhas de objeto replicados, enquanto o SDF se destina a detecção e diagnóstico de objetos de uma aplicação distribuída.

7.3. OGS (The Corba Object Group Service)

O ambiente OGS especifica uma arquitetura e um conjunto de interfaces para prover suporte a grupos sobre o ambiente CORBA [Fel98]. A abordagem adotada foi a especificação de serviços CORBA sobre um ORB padrão OMG [FGG97].

Na arquitetura OGS existem vários serviços que visam à comunicação confiável. Os serviços especificados são: o serviço de grupo (composto por um serviço multicast e um serviço membership que funcionam de forma complementar); o serviço de consenso que permite a vários objetos CORBA resolverem problemas de acordo em ambientes distribuídos; um serviço de mensagem para comunicação multicast confiável ponto a ponto; e um serviço de monitoração de objetos que provê mecanismos de detecção de falhas.

O serviço de grupo é o núcleo principal do ambiente OGS. O serviço membership gerencia os objetos do grupo, mantendo atualizada a lista de membros corretos através de suporte às entradas e saídas, notificações de visão e transferência de estado. O serviço multicast provê suporte para enviar invocações a todos os membros do grupo com confiabilidade e ordenação das mensagens.

O serviço de consenso, especificado na arquitetura OGS, é um serviço CORBA que permite ao conjunto dos objetos da aplicação tratar o problema de consenso em sistemas distribuídos. O serviço de consenso é usado no OGS para assegurar que mensagens enviadas por diferentes clientes sejam recebidas por todos os membros do grupo na mesma ordem. Este serviço é composto por duas categorias de objetos: os gerentes e os participantes de consenso. Os gerentes de consenso são objetos especificados no serviço que implementam o protocolo de consenso e chegam a um acordo entre eles. Eles atuam como caixas pretas, sendo sua implementação disponibilizada como serviço. O protocolo de consenso não é exposto à aplicação permitindo que este seja modificado sem impacto nos clientes do serviço. Os participantes de consenso são objetos da aplicação que propõem valores para os gerentes de consenso e recebem de volta o resultado. Eles são implementados pela aplicação, o que permite a adaptação do consenso para tarefas particulares.

O serviço de monitoração proposto na arquitetura OGS monitora objetos ao invés de processos e máquinas. Os componentes deste serviço são os próprios monitores (também chamados de detectores de defeitos), os objetos monitorados e os objetos notificados. Os monitores coletam informações sobre falhas dos objetos monitorados, baseando-se em mecanismos de *timeouts*. Eles mantêm uma lista com todos os objetos suspeitos de falhas. Os objetos monitorados são aqueles cuja falha deve ser detectada. Os objetos notificados registram-se no serviço de monitoração para receber, de forma assíncrona, notificações sobre falhas dos objetos monitorados. A primeira implementação do serviço de monitoração utilizava simples monitoria do tipo *pulling* sobre os objetos monitorados, enquanto que em uma segunda versão um modelo de monitoria *dual* é implementado. Neste modelo *dual*, uma combinação dos modelos de monitoria *pull* e *push* foi utilizada. Periodicamente, o objeto monitorado envia uma mensagem de *heartbeat* para informar que ele está ativo. Se um monitor não receber esta mensagem dentro do intervalo de tempo especificado, envia uma mensagem do tipo *pulling* para o objeto monitorado. Assim, nesta implementação do serviço de monitoração, mensagens para objetos monitorados acontecem quando estes não enviam um *heartbeat* dentro do tempo estipulado.

Na figura 7.3 podemos ver o detector de defeitos do serviço de monitoração monitorando os objetos OM1, OM2, OM3 e OM4. A figura representa a situação onde o objeto OM4 não enviou uma mensagem de *heartbeat* para o detector de defeitos no tempo esperado. O detector manda então uma mensagem do tipo *pull* para investigar se houve uma falha. Detectando o defeito do objeto OM4, os clientes inscritos no OGS são notificados, como pode ser visto na figura 7.3.

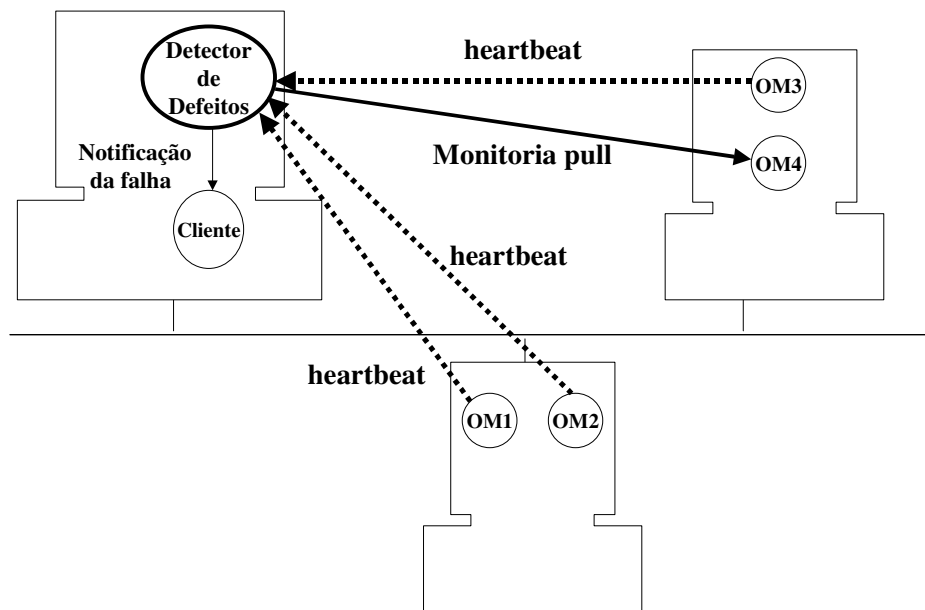


Figura 7.3 - Serviço de monitoração da arquitetura OGS

Em uma rede, um ou mais detectores de defeitos¹⁴ podem manter o estado dos objetos monitorados. Diferentes monitores podem fornecer diferentes informações a depender da sua localização na rede. O serviço de monitoração não garante ter um estado global consistente entre os módulos dos detectores de defeitos [FGS98]. Por exemplo, uma partição na rede pode levar a módulos detectores de defeitos, localizados em diferentes partições, fornecerem diferentes informações. Para atender questões de escalabilidade na monitoria de objetos pode ser usado um sistema hierárquico de monitoria [FGF99], onde detectores de defeitos enviam informações a monitores remotos em outras LANs (*local area network*). Neste caso, monitores localizados nas LANs podem se adaptar às características da rede.

Sobre a infraestrutura do OGS é possível especificar outros serviços, como por exemplo o Juggler [ME00]. Este é um serviço de gerenciamento de grupo para réplicas de objetos CORBA baseado no OGS. Pretende suportar automaticamente reconfiguração e recuperação de réplicas de objetos.

¹⁴ Redundância pode ser requerida por uma questão de tolerância a falhas.

7.3.1. Comparação do OGS com o SDF

A arquitetura do SDF apresenta aspectos semelhantes a arquitetura do OGS no que diz respeito aos serviço de monitoração e *membership*. Elas investem na modularidade proposta pelo padrão CORBA ao invés de propor soluções que modifiquem a semântica da especificação.

O objeto reconfiguração da arquitetura de diagnóstico do SDF tem atribuições correspondentes ao serviço de *membership* proposto no OGS, enquanto que os detectores de defeitos do SDF (agentes de conexão e de objetos) executam atividades que se assemelham às atividades do serviço de monitoração.

A arquitetura OGS tem por objetivo resolver problemas de comunicação em grupo. Por isso, o serviço de grupo compõe o núcleo principal deste trabalho, enquanto que o serviço de monitoração é um serviço complementar para atingir os propósitos de suporte a grupos de objetos. Já na arquitetura SDF, o foco é detecção e diagnóstico de falhas.

O serviço de monitoração no OGS é baseado em *timeouts*, com monitoria *pull* e *push* sobre objetos. Neste serviço não existe monitoria de máquinas. No SDF a monitoria acontece sobre máquinas, objetos e módulos do serviço, em um modelo de monitoria *pull*.

No serviço de monitoração não existe a garantia de um estado global consistente entre os detectores de defeitos, enquanto que no SDF, os módulos, que estão em contato com o servidor de nomes, mantém uma visão coerente sobre os processos monitorados.

7.4. Ferramenta Piranha

Piranha é uma ferramenta para monitoração e gerenciamento de disponibilidade de objetos em aplicações CORBA [Maf97]. Atua como um monitor da rede que reporta falhas através de uma interface gráfica. Nesta ferramenta, através do gerenciamento de disponibilidade é possível iniciar objetos que tenham falhado, fazer replicações e migrações. O usuário acompanha na interface gráfica a mudança de estados dos objetos monitorados. Para

executar estas atividades a ferramenta necessita de um suporte não disponível no ORB padrão OMG. Piranha roda sobre o Electra [Maf95], um ORB estendido desenvolvido pelo mesmo autor da ferramenta.

O Electra suporta abstrações de grupos de objetos, *multicast* confiável, transferência de estado e sincronismo virtual. É construído sobre subsistemas de comunicação de grupo tais como o Isis [BR94] ou Horus [RMB96] onde serviço de detecção pode fazer suposições de falhas que não estão corretas (é possível detectar como falho um objeto lento quando se utiliza apenas *timeouts*). Apesar disso, os resultados obtidos devem ser propagados consistentemente para todo o grupo. A preocupação, neste caso, é manter uma visão coerente entre os membros.

A figura 7.4 mostra a ferramenta Piranha sobre o ORB Electra. O Electra, como pode ser visto na figura, trabalha sobre uma plataforma Isis ou Horus.

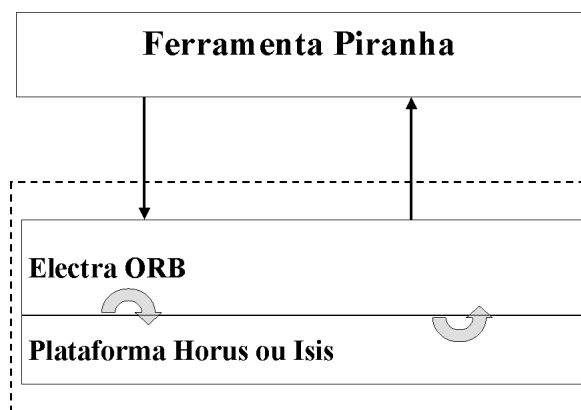


Figura 7.4 - Ferramenta Piranha sobre o Electra

7.4.1. Ferramenta Piranha e Visualizador SDF

Na arquitetura do SDF um Visualizador foi especificado e implementado para informar o estado dos processos e histórico das máquinas do grupo. O Visualizador obtêm estas informações dos serviços de detecção e reconfiguração da arquitetura SDF e disponibiliza para os usuários através de uma interface gráfica [item 3.7], semelhante à ferramenta

Piranha. No entanto, o foco do Visualizador se concentra nos aspectos de monitoração de falhas e acompanhamento do tempo de comunicação com os processos. Por sua vez, a ferramenta Piranha concentra-se em aspectos de monitoração de falhas e gerenciamento de disponibilidade de objetos. Este gerenciamento inclui atividades como replicação, migração e controle de localização dos objetos replicados.

Uma outra diferença entre o Visualizador e a ferramenta Piranha é que o Visualizador funciona sobre um ORB padrão OMG enquanto a ferramenta Piranha funciona sobre o Electra, que é um ORB estendido.

7.5. Comparação Final do SDF com os Trabalhos Correlatos Descritos

Muitos trabalhos tratam do problema de detecção de falhas em sistemas distribuídos. Analisamos, neste capítulo, alguns desses trabalhos que se propõem a executar, sobre plataforma CORBA, detecção e diagnóstico de falhas. Comparamos estes trabalhos correlatos com o serviço de diagnóstico de falhas proposto na dissertação.

Em resumo, podemos dizer que o SDF, diferente dos serviços DOORS e GroupPac, tem como objetivo tratar aspectos de detecção de falhas até o diagnóstico sem, todavia, atingir aspectos de replicação de componentes. Sobre a infraestrutura do SDF é possível especificar outros serviços assim como é feito com o OGS pelo serviço Juggler [ME00].

O Visualizador do SDF implementa funcionalidades semelhantes às apresentadas na ferramenta gráfica Piranha, no que diz respeito a monitoração e apresentação do estado dos objetos. A diferença entre eles está no gerenciamento de disponibilidade que a ferramenta Piranha executa e no acompanhamento do tempo de comunicação que é uma característica específica do Visualizador.

O SDF e os serviços DOORS monitoram falhas em três níveis: processo/objetos, máquinas e módulos de detecção. Os serviços GroupPac fazem monitoria de máquinas e de módulos detectores e o OGS faz monitoria de objetos. A monitoria do SDF e dos serviços GroupPac acontece segundo o modelo pull, enquanto no OGS e nos serviços DOORS esta monitoria é uma combinação dos modelos pull e push.

O SDF, o DOORS e o GroupPac, trabalham como serviços da arquitetura OMA sobre um ORB padrão OMG. Na ferramenta Piranha a abordagem foi de integração das funções de comunicação de grupo e detecção de falhas em um ORB não padrão, o Electra. Este ORB utiliza-se de subsistemas de comunicação em grupo, Isis ou Horus.

O SDF, diferente de todos os outros trabalhos descritos, implementa técnicas adicionais para detectar um processo como falho. Utilizamos, além de *timeouts*, consulta ao sistema operacional e testes locais sobre o processo suspeito.

No caso de suspeitas de falhas de máquinas um protocolo de acordo entre os módulos decide se os processos da máquina devem ser retirados do grupo. Distinguimos os diferentes tipos de falha que podem ocorrer com um processo, a saber: falha da máquina onde o processo reside, falha no próprio processo e congestionamento da infraestrutura de comunicação levando a altos tempos de comunicação. Todas estas informações podem ser acompanhadas pelos usuários da rede através do Visualizador. A distinção da real causa da falha do processo (se foi uma falha no próprio processo ou na máquina onde este reside) é uma característica bastante desejável em um serviço que se proponha a tratar questões de detecção em sistemas distribuídos. A informação de tempo de comunicação com um processo não é tratada em nenhum dos trabalhos correlatos analisados, pois para estes trabalhos o que importa é se o processo está falho ou se está ativo. No SDF, a informação de tempo de comunicação permite que a aplicação avalie as condições na infraestrutura de rede até o processo monitorado. Estas características, dentro de um ambiente com características assíncronas, fazem do SDF um serviço mais preciso no que diz respeito a detecção de falhas e na informação ao usuário do tipo de falha ocorrido.

Capítulo 8

Conclusão e Trabalhos Futuros

Finalizaremos a dissertação apresentando as conclusões e os trabalhos futuros.

8. Conclusão e Trabalhos Futuros

Esta dissertação apresentou uma arquitetura para diagnóstico de falhas de processos utilizados por uma aplicação distribuída. A arquitetura especifica um Serviço de Diagnóstico de Falhas (SDF) distribuído, sobre a plataforma CORBA, para monitorar processos e identificar problemas na execução destes. Este serviço foi implementado em Java, com alguns métodos nativos escritos na linguagem C, e testado em um ambiente JAVA/CORBA na rede de computadores do LaSiD/UFBA. O Serviço de Diagnóstico de Falhas foi concebido sobre os conceitos de gerenciamento, diagnóstico e detecção de falhas em sistemas distribuídos assíncronos.

Gerenciamento em sistemas distribuídos implica, entre outras coisas, em assegurar a execução correta da aplicação sobre a infraestrutura distribuída [VR01]. Para isso é necessário ter o diagnóstico dos processos e, considerando este diagnóstico, executar atividades que visam garantir a continuidade da aplicação, mesmo na presença de falhas. O diagnóstico envolve detectar as falhas que porventura aconteçam durante a execução da aplicação. O serviço de diagnóstico utiliza-se do mecanismo de CTI (Indicador de Tempo de Conectividade), proposto em [Mac00], para identificar com uma maior precisão a falha de um processo. O CTI foi implementado nas funções de detecção do serviço através do cálculo do tempo de comunicação entre os módulos do serviço e os processos monitorados, consultas ao sistema operacional, e, finalmente, através de testes locais aos processos suspeitos.

Geralmente, aplicações dependem de serviços diversos para o seu funcionamento, como por exemplo de um serviço WEB. Estes serviços também precisam ser monitorados para garantir o funcionamento da aplicação. Visando atender esta necessidade, especificamos um esquema de monitoria indireta através do uso de processos interfaces. Especificamos também, neste trabalho, um Visualizador para reportar, em uma interface gráfica, as informações de detecção e diagnóstico obtidas pelo SDF.

Realizamos alguns testes sobre o SDF em um ambiente com três, cinco e sete máquinas PC's 300 Mhz, 64 MB de memória, conectados por uma rede Ethernet 10 Mbps, sistema

operacional Windows NT Server 4.0, com software Visibroker 3.4 e JDK 1.2.2. Testamos situações com três, sete e quinze processos monitorados pelo serviço. Concluímos, com base nas amostras coletadas, que o aumento de processos no grupo não impactou de forma significativa no tempo de retirada de um processo falho do grupo, quando esta retirada se devia à falha de um processo individual. Esta é uma característica interessante que foi percebida durante os testes, pois nos leva a concluir que o serviço de diagnóstico se comporta de forma similar na retirada de um processo que falhou em um grupo com poucos processos ou com muitos, considerando a mesma quantidade de módulos SDF. Por outro lado, com base nas amostras coletadas, percebemos que o aumento do número de processos em uma máquina que se torne falha aumenta o tempo necessário para a reconfiguração do grupo. Também observamos que o aumento de módulos SDF no grupo aumenta o tempo de reconfiguração quando há falha de uma das máquinas.

Adequamos o serviço de diagnóstico para estar em conformidade com o padrão FT-CORBA no que se refere às atividades de detecção e diagnóstico de falhas. Assim, o SDF pode ser usado como uma infraestrutura de detecção e diagnóstico para que técnicas de tolerância a falhas, como replicação de componentes, possam ser implementadas em uma plataforma CORBA.

Analisamos alguns trabalhos que se propõem a realizar detecção e diagnóstico de falhas sobre plataforma CORBA. Percebemos que o SDF tem uma característica especial, quando comparado com estes trabalhos - a utilização de técnicas adicionais a *timeouts* para uma detecção mais precisa de falhas. Estas técnicas incluem : investigar o processo remoto no sistema operacional da máquina onde ele reside; realizar testes locais sobre o processo para responder a investigações sobre seu estado; e executar um protocolo de acordo para retirada de processos de máquinas falhas.

Realizamos também a identificação do tipo da falha. Esta identificação é uma informação importante para um serviço que trata de detecção e diagnóstico. Falhas de processos individuais, falhas de máquinas com processos monitorados, falhas de módulos do serviço e problemas na infraestrutura de comunicação podem ser identificados e reportados.

A medida que faz monitoria dos processos, o SDF apresenta o tempo de comunicação (*round trip*) encontrado com cada um destes. O cálculo deste tempo é disponibilizado para a

aplicação e permite que esta se adapte a diferentes níveis de qualidade de serviço. O acompanhamento do tempo de comunicação e a indicação de qual nível de qualidade de serviço está em vigor, são características impares do SDF.

Neste trabalho, relacionamos as áreas de gerenciamento, diagnóstico e detecção de falhas em sistemas distribuídos. Com os resultados obtidos, atingimos o objetivo proposto de desenvolver um serviço em CORBA que auxilie às aplicações distribuídas a diagnosticarem o estado de seus processos e serviços remotos. O serviço SDF contribui para atender as necessidades de gerenciamento de aplicações distribuídas a medida que fornece a infraestrutura para o desenvolvimento de aplicações de gerência e de replicação de componentes.

8.1. Trabalhos em andamento e propostas para trabalhos futuros

O módulo SDF é parte do projeto MONITOR (gerenciamento de sistemas distribuídos) em desenvolvimento no Laboratório de Sistemas Distribuídos – LaSiD/UFBA. Está em andamento no *LaSiD*, dando continuidade ao projeto, a implementação de uma ferramenta de gerenciamento e diagnóstico de processos sobre a infraestrutura do SDF.

O primeiro passo na construção desta ferramenta foi a migração do serviço de diagnóstico para a utilização do Visibroker 4.0 ao invés da versão 3.4 (disponível apenas para Irix, Digital Unix e Windows NT 4.0). A versão 4.0 do Visibroker está disponível para as plataformas Windows 2000/NT 4.0, Solaris 2.6/7.8, HP-UX 11.0, AIX 4.3.3 e Linux Red Hat 7.0. A migração envolveu algumas mudanças significativas no código do SDF, pois as classes de acesso ao servidor de nomes do Visibroker foram modificadas da versão 3.4 para a 4.0. Um novo objeto, *OperadorServicoNomes*, foi criado para centralizar todas as consultas ao servidor de nomes.

O Visualizador que apresenta o diagnóstico executado pelo SDF foi incorporado a esta ferramenta. A parte relativa ao gerenciamento de processos está sendo desenvolvida com a apresentação de opções para inicializar um processo, finalizar um processo existente,

modificar os valores de tempo de comunicação, finalizar todos os processos de uma máquina reiniciando em outra, entre outras funções.

Uma outra linha para continuação dos trabalhos do projeto MONITOR, envolve o desenvolvimento de um módulo genérico que faça acesso ao agente SNMP de uma máquina para saber o estado dos processos. Uma vez pronto, este módulo pode substituir o módulo existente (que utiliza objetos independentes para consultas ao sistema operacional), com a finalidade de investigar o estado dos processos em plataformas diversas. A proposta do projeto MONITOR é que o SDF e a ferramenta de gerenciamento estejam disponíveis para serem executados sobre um número expressivo de sistemas operacionais. Além da informação sobre o estado de processos, é possível obter outras informações da MIB para auxiliar no diagnóstico da aplicação.

Outra proposta de trabalho futuro envolve utilizar o serviço de diagnóstico para permitir a implementação de técnicas de replicação baseadas no padrão FT CORBA. Na versão atual do SDF isso não foi contemplado, pois replicação de objetos estava fora do escopo da dissertação. Dentro deste contexto, duas alternativas de trabalhos futuros podem ser desenvolvidas.

A primeira alternativa de trabalho consiste em possibilitar que o objeto gerente do SDF forneça os eventos de falhas detectadas por ele para um *replication manager* FT CORBA. Neste caso, o objeto gerente do SDF deverá implementar a interface *FaultNotifier*, podendo responder a qualquer *replication manager* que siga a especificação FT CORBA.

A figura 8.1 mostra esta alternativa para utilizar a infraestrutura de detecção de falhas do SDF para replicação de objetos. Através dos métodos da interface *FaultNotifier*, o *replication manager* FT CORBA, *fault analyzer* e *consumers* inscrevem-se no gerente SDF afim de receber as informações das falhas detectadas.

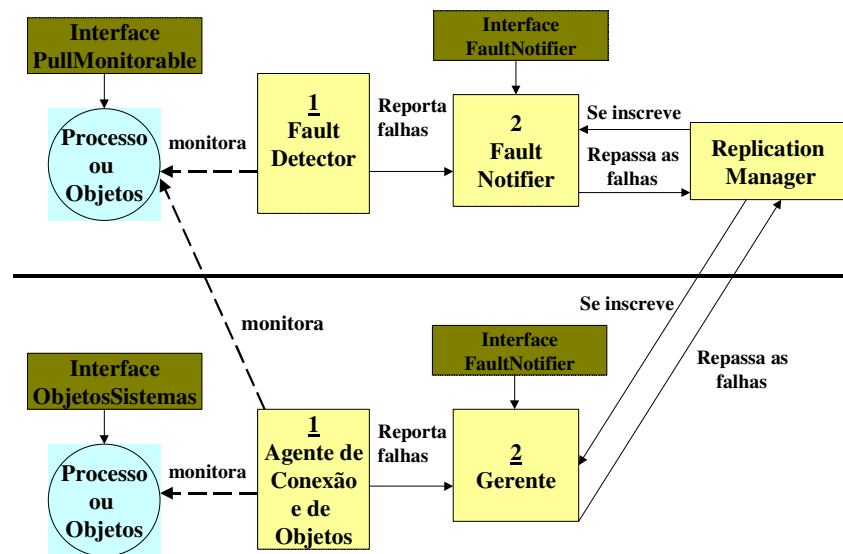


Figura 8.1 - Utilização da infraestrutura de detecção de falhas do SDF para replicação de objetos

A segunda alternativa é implementar um *replication manager* FT CORBA que se utilize dos serviços do objeto reconfiguração para a manutenção da lista de objetos do grupo. Nesta proposta, o *replication manager* recebe do objeto reconfiguração as informações de falhas que foram repassadas pelo objeto gerente. Neste caso, o *replication manager* exerce apenas as funções relativas ao controle de réplicas (com a criação de novos objetos quando acontecem falhas) e gerenciamento das propriedades do grupo. A interface *FaultNotifier* é implementada pelo objeto reconfiguração. Assim é possível utilizar a infraestrutura de detecção e *de diagnóstico* do SDF para implementar técnicas de replicação de objetos, como mostra a figura 8.2. Nesta figura vemos o objeto reconfiguração do SDF implementando a interface *FaultNotifier*. Um *replication manager* CORBA se inscreve no objeto reconfiguração para receber os eventos de falhas. Ele se utiliza dos serviços deste objeto com a finalidade de executar suas tarefas de replicação. Outros *consumers* também podem se inscrever para receber informações de detecção e diagnóstico.

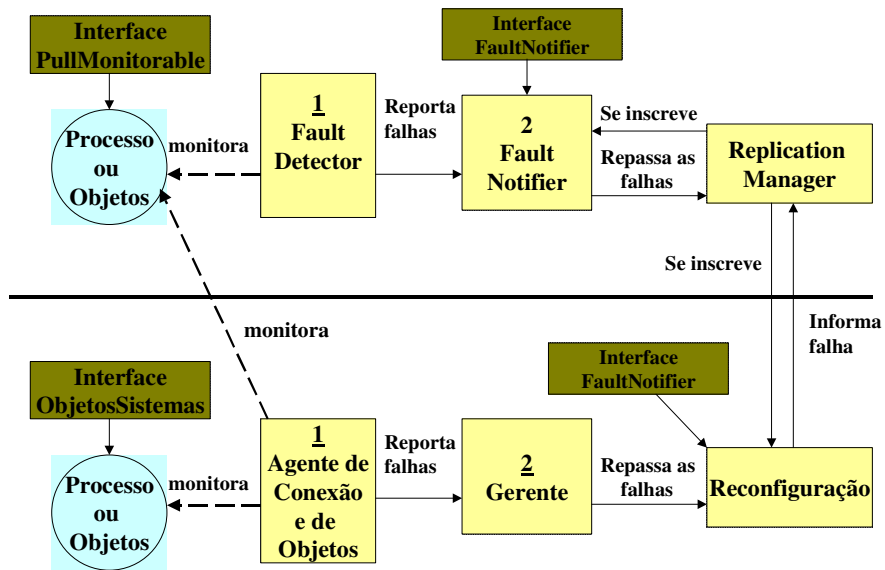


Figura 8.2 -Utilização da infraestrutura de detecção e *diagnóstico* de falhas do SDF para replicação de objetos

Um outro passo é investigar melhor o comportamento do SDF no acompanhamento de aplicações com muitos processos em várias máquinas. A utilização de domínios de tolerância a falhas é uma alternativa para escalabilidade da solução que precisa ser melhor estudada e testada.

Bibliografia

Bibliografia referenciada nos capítulos da dissertação

Bibliografia

- [Alm98] Almeida, Carlos., *Comunicação em sistemas quasi-síncronos: Suporte a aplicações dinâmicas de tempo-real*. Ph.D. dissertation, Universidade Técnica de Lisboa, Instituto Superior Técnico, Lisboa, Portugal, 1998.
- [AGP98] Abdallah, M., Guerraoui, R., Pucheral, P., “One-Phase Commit: Does It Make Sense?”. In *International Conference on Parallel and Distributed Systems*, pp 182-193, Taiwan, December 1998.
- [AV96] Almeida, Carlos., Veríssimo, Paulo., “Timing Failure Detection and Real-Time Group Communication in Quasi-Synchronous Systems”. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, pp.230-236, L'Aquila, Italy, June 1996.
- [Bat00] Batalha, M., “Arquitetura Orientada a Objetos para um Serviço Distribuído de Diagnóstico de Falhas sobre CORBA”, Relatório Técnico 0104/00, Laboratório de Sistemas Distribuídos, UFBA, Abril 2000.
- [BB92] Jr Bianchini, Ronald., Buskens, Richard., “Implementation of On-Line Distributed System-Level Diagnosis Theory”. In *IEEE Transactions on Computer*, V. 41, pp. 616-626., May 1992.
- [BHG87] Bernstein, P.A. , Hadzilacos, V., Goodmam, N., “*Concurrency Control and Recovery in Database Systems*” , 1ed., Addison Wesley, 1987.
- [BHM99] Badache, N., Hurfin M., Macêdo, R., “Solving The Consensus Problem in a Mobile Environment”, In *IEEE International Performance, Computing, and Communications Conference - IPCCC'99*, pp. 29-35, Phoenix/Scottsdale, Arizona, USA, February 1999.
- [Bir93] Birman, K., “The Process Group Approach to Reliable Distributed Computing”, *Communications of ACM*, v. 9, n.12, pp. 36-53, December 1993.
- [BM01] Batalha, M., Macêdo, R., “Um serviço tolerante a falhas para o gerenciamento de Sistemas Distribuídos sobre CORBA”, Aceito em *XXVII Latin-American Conference on Informatics - CLEI'2001*, Mérida, Venezuela, Setembro 2001.
- [BR94] Birman, Kenneth P., Renesse, Robert Van., “Reliable Distributed Computing with the ISIS Toolkit”. In : *IEEE Computer Society Press*. Los Alamitos, California, 1994.
- [CDK96] Coulouris, George., Dollimore, Jean., Kindberg, Tim., “*Distributed Systems – Concepts and Design*”. 2 ed. , London, Addison-Wesley, 1996.
- [CHY98] Chung, P. Emerald., Hung, Yennun., Yajnik, Shalini., “DOORS: Providing Fault Tolerance for CORBA Applications”, poster at *Middleware 98*, <http://www.bell-labs.com/org/11356/docs/doors97.doc>.

- [Cir99] Cirne, L., 1999, *Tolerância a Falhas em JAVA Através de Comunicação em Grupo*, Tese de M.sc., COPIN/UFPe, Campina Grande, PB, Brasil.
- [CGS00] Charron-Bost, B., Guerraoui, R., Schiper, Andre., "Synchronous System and Perfect Failure Detector: Solvability and Efficiency Issues". In *International Conference on Dependable Systems and Networks (DSN 2000)*, pp 523-533, New York, New York, June 2000.
- [CT96] Chandra, T. and Toueg, S., "Unreliable Failure Detectors for Realiable Distributed Systems", *Journal of the ACM*, v.43 n.1: pp.225-267, March 1996.
- [DBA99] Duarte, E., Brawerman, A., Albin, L., "Um Algoritmo para Diagnóstico Distribuído com Informações Datadas", In *17º Simpósio Brasileiro de Redes de Computadores*, pp 501-515, Salvador, maio 1999.
- [DN98] Duarte Jr, L., Nanya, T., "A hierarchical Adaptative Dist. System-level Diagnosis Algorithm". In *IEEE Trans. On Computers*, v. 47, n.1, pp 34-45, January 1998.
- [FC96] Fetzer, C., Cristian, F., "Fail-aware failure detectors". In *Proceedings of the 15th Symposium on Reliable Distributed Systems*, pp. 200-210, Niagara-on-the-Lake, Canada, October 1996.
- [FC98] Cristian, F., Fetzer, C., "The Timed Asynchronous System Model", In *28th Annual International Symposium on Fault-Tolerant Computing*, pp140-150, Munich, Germany, june, 1998
- [Fel98] Felber, Pascal. "The CORBA Object Grup Service – A service approach to Object Groups in CORBA", Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, EPFL 1998.
- [FGF99] Felber, P., Guerraoui, R., Fayad, M., "Putting OO distributed programming to work". In *Communications of the ACM*, v. 42, pp 97-101, January 1999.
- [FGG96] Felber, Pascal., Garbinato, Benoît., Guerraoui, Rachid. "The Design of a CORBA Group Communication Service". In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, pp. 150-160, Niagara-on-the-lake, Canada, October 1996.
- [FGG97] Felber, P., Garbinato, B., Guerraoui, R., *Special Issues in Object-Oriented Programming*, chapter Towards Realiable CORBA: Integracion vs. Service Approach. pp. 199-205, dpunkt-Verlag, 1997.
- [FGS98] Felber, P., Guerraoui, R., Schiper, A. "The implementation of a CORBA group communication service". In *Theory and Practice of Object Systems*, v.4, n.2, pp. 93-105, 1998.
- [Fla91] Flaviu, C., "Reaching Agreement on Processor-Group Memebership in Synchronous Distributed Systems". In *Distributed Computing*, v.4, pp.175-187, 1991.

- [FLP85] Ficher, M. J., Lynch, N. A., Paterson, M. S. "Impossibility of Distributed Consensus with One Faulty Process". In *Journal of the ACM*, v. 32, n.2, pp. 374-382, April 1985.
- [FM96] Fonseca, J., Martins, G., "Curso de estatística". 6ed., São Paulo, Atlas S.A, 1996.
- [Gar99] Gartner, Felix. C., "Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments", In *ACM Computing Surveys*, v.31, n.1, March 1999.
- [GLS96] Guerraoui, R., Larrea, M., Schiper, A., "Reducing the Cost for Non-Blocking in Atomic Commitment ", In *16th International Conference on Distributed Computing Systems (ICDCS '96)*, pp. 692-698, Hong Kong, May 1996.
- [GS95] Guerraoui, R., Schiper, A. , "The Decentralized Non-Blocking Atomic Commitment Protocol", In *7th IEEE Symposium on Parallel and Distributed Processing (SPDP '95)*, pp 2-10 , San Antonio, Texas, October 1995.
- [HMRT99] Hurfin, M., Macêdo, R., Raynal, M., Tronel, F., "A General Framework to Solve Agreement Problems". In *18th IEEE Symposium on Reliable Distributed Systems*, pp. 56-66, Lausanne, Switzerland, October, 1999.
- [Jal94] Jalote, Pankaj. "*Fault Tolerance in Distributed Systems*". Englewood Cliffs, New Jersey, PTR Prentice Hall, 1994.
- [Kea97] Keahey, K. *A Brief Tutorial on CORBA*, in CORBA for Beginners, Object Management Group, 1997.
- [LAF99] Larrea M., Arevalo S., Fernandez A., "Efficient Algorithms To Implements Unreliable Failure Detectors in Partially Synchronous Systems". In *Proceedings of the 13th Symposium on Distributed Computing (DISC'99)*, pp. 34-48, Bratislava (Slovakia), September 1999.
- [Lap85] Laprie, J.C., "Dependable Computing and Fault Tolerance: Concepts and Terminology.", In *15th International Symposium on Fault Tolerant Computing Systems*, pp.2-11, Ann Arbor, Michigan, June 1985.
- [LF00] Lung, Lau., Fraga, Joni., "Detecção de Falhas para Redes de Larga Escala no Fault Tolerant CORBA", In *Anais do II Workshop de Testes e Tolerância a Falhas - WTF'2000*, pp. 10-15, Curitiba, Paraná, Julho 2000.
- [LFFOR99] Lung, Lau. , Fraga, Joni. , Farines, Jean-Marie, Ogg, Michael. , Ricciardi, Aleta. "CosNamingFT - A Fault-Tolerant CORBA Naming Service". In *Proceeding of the 18th International Symposium on Reliable Distributed Systems - SRDS'99*, pp. 254-263, Lausanne, Switzerland, October 1999.
- [LFFO00] Lung, Lau., Fraga, Joni., Farines, Jean-Marie., Oliveira, Jorge Ricardo S., "Experiências com Comunicação de Grupo nas Especificações Fault Tolerant CORBA", In *Anais do XVIII Simpósio Brasileiro de Redes de Computadores - SBRC'2000*, pp. 85-100, Belo Horizonte, Minas Gerais, Maio 2000.

- [LFPS01] Lung, Lau., Fraga, Joni., Padilha, Ricardo., Souza, Luciana., “Adaptando as Especificações FT-CORBA para Redes de Larga Escala”, In *Anais do XIX Simpósio Brasileiro de Redes de Computadores - SBRC'2001*, Florianópolis, Santa Catarina, maio de 2001.
- [Mac94] Macedo, Raimundo. “Fault-Tolerant Group Communication Prorocols for Asynchronous Systems”. Ph.D. dissertation, University of Newcastle upon Tyne, Department of Computing Science, November 1994.
- [Mac00] Macedo, Raimundo. “Failure Detection in Asynchronous Distributed Systems”, In *II Workshop on Test and Fault-Tolerance*, pp. 76-81, Curitiba, Paraná, July 2000.
- [Maf95] Silvano Maffeis. *Run-Time Support for Object-Oriented Distributed Programming*. Ph.D. dissertation, University of Zurich, Zurich, 1995.
- [Maf97] Maffeis, Silvano., “Piranha: A CORBA Tool for High Availability”. In: *Computer*, pp. 56-66, v.30, n.4, April 1997.
- [ME00] Mora, Marcos., Endler, Markus., “Juggler: A Management Service for CORBA Object Groups”, In *Anais do XVIII Simpósio Brasileiro de Redes de Computadores - SBRC'2000*, pp. 227-242, Belo Horizonte, Minas Gerais, Maio de 2000.
- [NGYS00] Natarajan, Balachandran., Gokhale, Aniruddha., Yajnik, Shalini., Schmidt, Douglas., “DOORS: Towards High-Performance Fault Tolerant CORBA”, In *Proceedings of the International Symposium on Distributed Objects and Applications*, pp.39-49, Antwerp, Belgium, September 2000.
- [OMG96] Object Management Group. “The Common Object Request Broker 2.0/IIOP Specification”. Revision 2.0, OMG Document 1996-08-04, 1996.
- [OMG00] Object Management Group. “Fault Tolerant CORBA Specification, V1.0”. OMG Document 2000-04-04, 2000.
- [RMB96] Renesse, Robert Van., Maffeis, Silvano., Birman, Kenneth P., “Horus: A Flexible Group Communication System”. In *Communications of the ACM*. Pp. 76-83, April, 1996.
- [RD95] Rangarajan, Sampath., Dahbura, A.T.”A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies”, In *IEEE Transactions on Computers*, pp. 312-334, v. 44, n.2, February, 1995.
- [Sch97] Schmidt, D., “*Overview of CORBA*”, In *Corba for Beginners Object Management Group*, 1997.
- [SNCKR98] Shaffer, R., Nelson, Edward., Chico, N., Korey, J., Nelson, Elizabeth., Ross, J., “SPSS for Windows, Version 9: A Brief Tutorial”, McGraw-Hill, 1998. <http://www.csubak.edu/ssric/Modules/SPSS/SPSFirst.htm>

- [Spi93] Spiegel, R. Murray., “Estatística”, 3º Edição, Editora Afiliada, São Paulo, Makron Books, 1993.
- [TC00] Chen, Wei., Toueg, Sam., “On the Quality of Service of Failure Detectors”. In *International Conference on Dependable Systems and Networks (DSN 2000)*, pp. 191-201, New York, New York, June 2000.
- [YD96] Yang, Z and Duddy, K. *CORBA: A Platform for Distributed Object Computing*. Report of Distributed Systems Group. Information System Institute, Technical University of Vienna. 1996.
- [VA95] Veríssimo, Paulo., Almeida, Carlos., “Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models “ In *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, v.7, n.4, pp.35-39, 1995.
- [Ver97] Veríssimo, Paulo., “On the Role of Time in Distributed Systems”, In *6th IEEE Workshop on Future Trends os Distributed Computing Systems (FTDCS'97)*, pp. 316-322, Tunis, Tunísia, October 1997.
- [VR01] Veríssimo, Paulo., Rodrigues, Luís. “*Distributed Systems for System Architects*”, 1 ed., Kluwer Academic Publishers, 2001.
- [X.701] ITU-T Recommendation X.701, Systems Management Overview, 1997.

Anexos

Em anexos apresentamos tabelas extraídas do SPSS nos testes executados sobre o serviço de diagnóstico de falhas.

Tabelas resultantes dos testes sobre o serviço de diagnóstico.

Todas as tabelas deste anexo foram geradas pelo SPSS.

Na tabela **Ranks**, em *Amostra* temos o número da amostra. O parâmetro *N* representa o número de valores válidos em cada amostra depois de executadas as exclusões dos extremos, conhecidos como *outliers*. O *Mean Rank* significa o valor de posto médio da amostra que foi utilizado nas formulas dos testes. *Sum of Ranks* é o somatório dos postos médios que aparece quando esta tabela é gerada nos testes de Mann-Whitney.

Alguns dados gerados nas tabelas não foram utilizados nas nossas análises pois indicavam valores de testes e outras distribuições não-paramétrica de probabilidade, como por exemplo os valores encontrados de *Chi-Square*, *Wilcoxonw*, etc.

Os parâmetros que nos interessam nas tabelas a seguir são os valores das amostras e o **Asymp. Sig.** que indica o valor de erro tolerável¹⁵. Se o valor encontrado de **Asymp. Sig.** for menor do que 0.05 dizemos que as amostras apresentam diferenças nas médias dos tempos de reconfiguração coletados.

Tabela A – Teste de Kruskal-Wallis aplicado sobre as amostras 1, 2 e 3 (3 máquinas com 3, 7 e 15 processos)

	AMOSTRA	N	Mean Rank
Tempo(ms)	1.00	8	13.25
	2.00	8	15.44
	3.00	8	8.81
	Total	24	

	Tempo(ms)
Chi-Square	3.902
df	2
Asymp. Sig.	.142

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

A tabelas A apresenta os resultados do teste de Kruskal-Wallis aplicado sobre as amostras 1, 2 e 3 (3 máquinas com 3, 7 e 15 processos). O teste indicou que não há uma diferença significativa nas médias dos tempos de reconfiguração entre as amostras 1, 2 e 3.

¹⁵ Consideramos, em nossos testes, um valor de erro tolerável é de 5%.

Tabela B - Teste de Kruskal-Wallis aplicado sobre as amostras 4, 5 e 6 (5 máquinas com 3, 7 e 15 processos)

Ranks			
	AMOSTRA	N	Mean Rank
Tempo(ms)	4.00	9	15.28
	5.00	9	12.61
	6.00	9	14.11
	Total	27	

Test Statistics ^{a,b}	
	Tempo(ms)
Chi-Square	.541
df	2
Asymp. Sig.	.763

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

A tabela B apresenta o resultado do teste de Kruskal-Wallis aplicado sobre as amostras 4, 5 e 6 (5 máquinas com 3, 7 e 15 processos). O valor de Asymp. Sig. encontrado indica que não existe uma diferença significativa entre as médias dos valores de tempo de reconfiguração.

Tabela C - Teste de Kruskal-Wallis aplicado sobre as amostras 7, 8 e 9 (7 máquinas com 3, 7 e 15 processos)

Ranks			
	AMOSTRA	N	Mean Rank
Tempo(ms)	7.00	9	11.89
	8.00	7	12.43
	9.00	9	14.56
	Total	25	

Test Statistics ^{a,b}	
	Tempo(ms)
Chi-Square	.717
df	2
Asymp. Sig.	.699

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

O resultado do teste de Kruskal-Wallis aplicado sobre as amostras 7, 8 e 9 (7 máquinas com 3, 7 e 15 processos) pode ser visto na tabela C. O resultado encontrado de Asymp. Sig. mostra que não é significativa as diferenças entre as médias dos valores de tempo de reconfiguração.

Tabela D - Teste de Kruskal-Wallis aplicado sobre as amostras 1, 4 e 7 (3 Processos em 3, 5 e 7 máquinas)

Ranks			
	AMOSTRA	N	Mean Rank
Tempo(ms)	1.00	8	7.50
	4.00	9	18.22
	7.00	9	14.11
	Total	26	

Test Statistics ^{a,b}	
	Tempo(ms)
Chi-Square	9.109
df	2
Asymp. Sig.	.011

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

O resultado do teste de Kruskal-Wallis aplicado sobre as amostras 1, 4 e 7 (3 Processos em 3, 5 e 7 máquinas) indicou que existe uma diferença nas médias dos tempos de reconfiguração, conforme parâmetro Asymp. Sig = 0.011 encontrado na tabela D.

Tabela E - Teste de Kruskal-Wallis aplicado sobre as amostras 2, 5 e 8 (7 Processos em 3, 5 e 7 máquinas)

Ranks			
	AMOSTRA	N	Mean Rank
Tempo(ms)	2.00	8	10.88
	5.00	9	12.72
	8.00	7	14.07
	Total	24	

Test Statistics ^{a,b}	
	Tempo(ms)
Chi-Square	.804
df	2
Asymp. Sig.	.669

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

O teste de Kruskal-Wallis aplicado sobre as amostras 2, 5 e 8 (7 Processos em 3, 5 e 7 máquinas) pode ser visto na tabela E. O resultado deste teste indicou que não existe uma diferença nas médias dos tempos de reconfiguração, conforme parâmetro Asymp. Sig = 0.669 encontrado na tabela E.

Tabela F - Teste de Kruskal-Wallis aplicado sobre as amostras 3, 6 e 9 (15 Processos em 3, 5 e 7 máquinas)

	AMOSTRA	N	Mean Rank
Tempo(ms)	3.00	8	6.75
	6.00	9	16.17
	9.00	9	16.83
	Total	26	

	Tempo(ms)
Chi-Square	9.641
df	2
Asymp. Sig.	.008

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

O resultado do teste de Kruskal-Wallis aplicado sobre as amostras 3, 6 e 9 (15 Processos em 3, 5 e 7 máquinas) indicou que existe uma diferença nas médias dos tempos de reconfiguração.

Tabela G – Teste de Kruskal-Wallis aplicado sobre as amostras 1, 2 e 3 da Tabela 4 (3 máquinas com 1, 3 e 6 processos)

	AMOSTRA	N	Mean Rank
TEMPO	1.00	10	8.35
	2.00	10	13.25
	3.00	10	24.90
	Total	30	

	TEMPO
Chi-Square	19.171
df	2
Asymp. Sig.	.000

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

A tabela G apresenta o resultado do teste de Kruskal-Wallis aplicado sobre as amostras 1, 2 e 3 da Tabela 4 (3 máquinas com 1, 3 e 6 processos). O resultado de Asymp. Sig. indica uma diferença significativa nas médias dos tempos de reconfiguração.

Tabela H – Teste de Kruskal-Wallis aplicado sobre as amostras 4, 5 e 6 da Tabela 4 (5 máquinas com 1, 3 e 6 processos)

	AMOSTRA	N	Mean Rank
TEMPO	4.00	10	5.55
	5.00	10	15.45
	6.00	9	25.00
	Total	29	

	TEMPO
Chi-Square	25.488
df	2
Asymp. Sig.	.000

a. Kruskal Wallis Test

b. Grouping Variable: AMOSTRA

A tabela H apresenta o resultado do teste de Kruskal-Wallis aplicado sobre as amostras 4, 5 e 6 da Tabela 4 (5 máquinas com 1, 3 e 6 processos). O valor de Asymp. Sig indica um alto nível de significância, ou seja uma diferença nas médias dos tempos de reconfiguração das amostras.

Tabela I – Teste de Mann-Whitney aplicado sobre as amostras 1,2 da Tabela 4 (3 máquinas com 1, 3 processos)

	AMOSTRA	N	Mean Rank	Sum of Ranks
TEMPO	1.00	10	8.00	80.00
	2.00	10	13.00	130.00
	Total	20		

	TEMPO
Mann-Whitney U	25.000
Wilcoxon W	80.000
Z	-1.950
Asymp. Sig. (2-tailed)	.051
Exact Sig. [2*(1-tailed Sig.)]	.063 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

O teste de Mann-Whitney aplicado sobre as amostras 1, 2 da Tabela 4 (3 máquinas com 1, 3 processos) indica um valor limite do nível de significância, que é de 0.05.

Tabela J – Teste de Mann-Whitney aplicado sobre as amostras 2, 3 da Tabela 4 (3 máquinas com 3, 6 processos)

Ranks				Test Statistics ^b		
	AMOSTRA	N	Mean Rank	Sum of Ranks	TEMPO	
TEMPO	2.00	10	5.75	57.50	Mann-Whitney U	2.500
	3.00	10	15.25	152.50	Wilcoxon W	57.500
	Total	20			Z	-3.682
					Asymp. Sig. (2-tailed)	.000
					Exact Sig. [2*(1-tailed Sig.)]	.000 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

O resultado do teste de Mann-Whitney aplicado sobre as amostras 2, 3 da Tabela 4 (3 máquinas com 3, 6 processos) pode ser visto na tabela J. O valor encontrado de Asymp. Sig. indica que existe uma diferença significativa nas médias dos valores de tempo de reconfiguração das amostras.

Tabela K – Teste de Mann-Whitney aplicado sobre as amostras 1,3 da Tabela 4 (3 máquinas com 1, 6 processos)

Ranks				Test Statistics ^b		
	AMOSTRA	N	Mean Rank	Sum of Ranks	TEMPO	
TEMPO	1.00	10	5.85	58.50	Mann-Whitney U	3.500
	3.00	10	15.15	151.50	Wilcoxon W	58.500
	Total	20			Z	-3.591
					Asymp. Sig. (2-tailed)	.000
					Exact Sig. [2*(1-tailed Sig.)]	.000 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

O resultado do teste de Mann-Whitney aplicado sobre as amostras 1, 3 da Tabela 4 (3 máquinas com 1, 6 processos) pode ser visto na tabela K. É possível concluir, com base nos valores de Asymp. Sig., que há uma diferença nas médias dos valores de tempo de reconfiguração das amostras.

Tabela L – Teste de Mann-Whitney aplicado sobre as amostras 4,5 da Tabela 4 (5 máquinas com 1, 3 processos)

Ranks				Test Statistics ^b	
AMOSTRA	N	Mean Rank	Sum of Ranks		TEMPO
TEMPO 4.00	10	5.55	55.50	Mann-Whitney U	.500
5.00	10	15.45	154.50	Wilcoxon W	55.500
Total	20			Z	-3.885
				Asymp. Sig. (2-tailed)	.000
				Exact Sig. [2*(1-tailed Sig.)]	.000 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

Os resultados dos testes de Mann-Whitney aplicado sobre as amostras 4, 5 da Tabela 4 (5 máquinas com 1, 3 processos) podem ser visto na Tabela L. O valor encontrado de Asymp. Sig. indica que existe uma diferença nas médias dos valores de tempo de reconfiguração das amostras.

Tabela M – Teste de Mann-Whitney aplicado sobre as amostras 5,6 da Tabela 4 (5 máquinas com 3, 6 processos)

Ranks				Test Statistics ^b	
AMOSTRA	N	Mean Rank	Sum of Ranks		TEMPO
TEMPO 5.00	10	5.50	55.00	Mann-Whitney U	.000
6.00	9	15.00	135.00	Wilcoxon W	55.000
Total	19			Z	-3.803
				Asymp. Sig. (2-tailed)	.000
				Exact Sig. [2*(1-tailed Sig.)]	.000 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

A tabela M apresenta os valores encontrados no teste de Mann-Whitney aplicado sobre as amostras 5, 6 da Tabela 4 (5 máquinas com 3, 6 processos). É possível concluir, com base nos valores de Asymp. Sig., que há uma diferença nas médias dos valores de tempo de reconfiguração das amostras.

Tabela N – Teste de Mann-Whitney aplicado sobre as amostras 1, 3 da Tabela 4 (1 Processo com 3 e 5 máquinas)

Ranks				Test Statistics ^b	
	AMOSTRA	N	Mean Rank	Sum of Ranks	
TEMPO	1.00	10	7.15	71.50	Mann-Whitney U
	4.00	10	13.85	138.50	Wilcoxon W
	Total	20			Z
					Asymp. Sig. (2-tailed)
					Exact Sig. [2*(1-tailed Sig.)]

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

Os resultados do teste de Mann-Whitney aplicado sobre as amostras 1, 3 da Tabela 4 (1 Processo com 3 e 5 máquinas) pode ser visto na tabela N. O valor encontrado de Asymp. Sig. indica que existe uma diferença nas médias dos valores de tempo de reconfiguração das amostras 1 e 4 .

Tabela O – Teste de Mann-Whitney aplicado sobre as amostras 2, 5 da Tabela 4 (3 Processos com 3 e 5 máquinas)

Ranks				Test Statistics ^b	
	AMOSTRA	N	Mean Rank	Sum of Ranks	
TEMPO	2.00	10	5.50	55.00	Mann-Whitney U
	5.00	10	15.50	155.00	Wilcoxon W
	Total	20			Z
					Asymp. Sig. (2-tailed)
					Exact Sig. [2*(1-tailed Sig.)]

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

A tabela O apresenta os resultados do teste de Mann-Whitney aplicado sobre as amostras 2, 5 da Tabela 4 (3 Processos com 3 e 5 máquinas). Podemos concluir, com base no valor de Asymp. Sig. que as médias das amostras tem diferenças significativas.

Tabela P – Teste de Mann-Whitney aplicado sobre as amostras 3, 6 da Tabela 4 (6 Processos com 3 e 5 máquinas)

Ranks				Test Statistics ^b	
AMOSTRA	N	Mean Rank	Sum of Ranks		TEMPO
TEMPO 2.00	10	5.50	55.00	Mann-Whitney U	.000
5.00	10	15.50	155.00	Wilcoxon W	55.000
Total	20			Z	-3.922
				Asymp. Sig. (2-tailed)	.000
				Exact Sig. [2*(1-tailed Sig.)]	.000 ^a

a. Not corrected for ties.

b. Grouping Variable: AMOSTRA

Os resultados do teste de Mann-Whitney aplicado sobre as amostras 3, 6 da Tabela 4 (6 Processos com 3 e 5 máquinas) pode ser visto na tabela P. O valor de Asymp. Sig indica que existe diferença significativa nas médias dos valores de tempo de reconfiguração.