Pontifícia Universidade Católica
do Rio de Janeiro

Adolfo Guilherme Silva Correia

Uma arquitetura baseada em agentes de software para a automação de processos de gerência de falhas em redes de telecomunicações

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio

Orientador: Prof. Carlos José Pereira de Lucena

Rio de Janeiro
Abril de 2007

Pontifícia Universidade Católica
do Rio de Janeiro

Adolfo Guilherme Silva Correia

Uma arquitetura baseada em agentes de software para a automação de processos de gerência de falhas em redes de telecomunicações

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Carlos José Pereira de Lucena
Orientador
Departamento de Informática — PUC-Rio

Prof. Sérgio Colcher
Departamento de Informática — PUC-Rio

Prof. Ricardo Choren Noya
Seção de Engenharia de Computação — IME

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico
PUC-Rio

Rio de Janeiro, 03 de Abril de 2007

Adolfo Guilherme Silva Correia

Concluiu o curso de Engenharia de Computação pelo IME (Instituto Militar de Engenharia) em 2003. Atua na linha de pesquisa de Engenharia de Software e de Gerência de Redes e no desenvolvimento de sistemas para gerência de redes.

Ficha Catalográfica

Correia, Adolfo Guilherme Silva

Uma arquitetura baseada em agentes de software para a automação de processos de gerência de falhas em redes de telecomunicações / Adolfo Guilherme Silva Correia; orientador: Carlos José Pereira de Lucena. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2007.

114 f: il. ; 30 cm

Dissertação (Mestrado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Teses. 2. Sistemas Multi-Agentes. 3. Gerência de Redes. I. Lucena, Carlos José Pereira de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Para minha família.

Agradecimentos

À minha família, que sempre me incentivou.
Ao meu orientador Prof. Lucena, pelo apoio e orientação.
Ao Prof. Casanova, pelo conselho oportuno em momento de desânimo.
À Capes, pelo apoio financeiro.
Aos colegas do Departamento de Informática.
Aos colegas Edgar, Fábio e Leonardo, amigos e incentivadores.
Aos colegas Carlo, Carlos Augusto e Levy.

Resumo

Correia, Adolfo Guilherme Silva; Lucena, Carlos José Pereira de. **Uma arquitetura baseada em agentes de software para a automação de processos de gerência de falhas em redes de telecomunicações**. Rio de Janeiro, 2007. 114p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os últimos anos têm sido marcados pelo significativo crescimento em todo o mundo da demanda por serviços de telecomunicações. Tal cenário de expansão de redes e da necessidade de coexistência e interoperabilidade de diferentes tecnologias de forma economicamente viável proporciona grandes desafios para a gerência, operação e manutenção de redes de telecomunicações.

O presente trabalho apresenta alguns dos principais modelos e paradigmas de gerência de redes tradicionalmente empregados em redes de telecomunicações e que ainda hoje são amplamente utilizados pela indústria. Muitos dos modelos apresentados foram significativamente influenciados por conceitos e técnicas oriundos da área de engenharia de software. Uma grande ênfase é dada particularmente ao uso de técnicas baseadas em agentes de software para gerência de redes. Para tanto, importantes conceitos sobre agentes de software são apresentados, assim como exemplos de trabalhos em que agentes de software são utilizados no domínio de gerência de redes.

Por fim, é proposta uma arquitetura baseada em agentes de software para gerência de falhas em redes legadas de telecomunicações, que são comumente gerenciadas por sistemas centralizados. O objetivo principal desta arquitetura é permitir o diagnóstico e a correção de falhas de rede de forma a não sobrecarregar o sistema centralizado de gerência. Para tanto, são utilizados agentes de software que distribuem informações mantidas no sistema centralizado para outros agentes do sistema. Desta forma, é possível que os agentes responsáveis por executar os procedimentos de diagnóstico e correção de falhas desempenhem suas atividades sem a necessidade de uma comunicação direta com o sistema centralizado.

Palavras–chave

Sistemas Multi-Agentes. Gerência de Redes.

Abstract

Correia, Adolfo Guilherme Silva; Lucena, Carlos José Pereira de. **A software agents based architecture for the automation of fault management processes in telecommunications networks**. Rio de Janeiro, 2007. 114p. MsC Thesis — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The last few years have been marked by a significant and worldwide growth in the demand for telecommunications services. Such scenery of network expansion and the need for coexistence and interoperability of different technologies in an economically viable way provides great challenges for the management, operation and maintenance of telecommunications networks.

This work presents some of the main network management models and paradigms traditionally employed in telecommunications networks and that still count with wide adoption in the industry as of this day. Many of the presented models have been significantly influenced by concepts and techniques originated in the software engineering field. A great emphasis is particularly given to the use of network management techniques based on software agents. To this end, important concepts of software agents are presented, as well as examples of works where software agents are used in the network management domain.

Finally, an architecture based on software agents used for fault management in legacy telecommunications networks, which are usually managed by centralized systems, is proposed. The main objective of this architecture is to allow the diagnosis and the correction of network faults in a way not to overload the centralized management system. To this end, the architecture uses software agents that distribute information maintained in the centralized management system to other agents of the system. In such way, it is possible for the agents responsible for executing the fault diagnosis and correction procedures to perform their activities without the necessity for direct communication with the centralized system.

Keywords

Multi-Agents Systems. Network Management.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

1.1
Motivação

Os últimos anos foram marcados pelo crescimento da demanda por serviços de telecomunicações, tanto de voz quanto de dados, em todo o mundo, especialmente devido à popularização da Internet e ao fenômeno da globalização. Como resposta a esta crescente demanda, companhias do setor de telecomunicações e instituições governamentais vêm instalando a infra-estrutura necessária para o tráfego de voz, vídeo e dados, viabilizando a comunicação entre pontos remotos. Os principais componentes desta infra-estrutura são as redes de telecomunicações compostas por equipamentos como roteadores e transmissores e pelas conexões entre estes equipamentos como cabos de fibra ótica e enlaces de rádio, por exemplo.

Essas redes de equipamentos de telecomunicações têm se tornado cada vez maiores e mais complexas. As dimensões que elas têm tomado tornam cada vez mais difícil satisfazer os requisitos de confiabilidade, desempenho e segurança necessários para o seu bom funcionamento. Além disso, a operação e a manutenção destas redes ainda são dificultadas devido à necessidade da convivência de diferentes protocolos e tecnologias, da integração de redes legadas, da heterogeneidade de fabricantes, equipamentos e sistemas de gerência, e da ocasional não conformidade com os padrões existentes. Estes desafios já vêm sendo registrados há mais de dez anos, como pode ser observado em [Voruganti94], por exemplo, em que se apresentam esforços de organismos de normalização no sentido de amenizar problemas de interoperabilidade. No entanto, a experiência mostra que estas questões ainda hoje causam transtornos na operação de redes de telecomunicações.

1.2
Gerência de Redes

Diante deste cenário de aumento de dimensão e de complexidade das redes de telecomunicações, torna-se cada vez mais necessária a criação (ou adequação) de modelos e estratégias de gerência de redes que permitam a automatização dos processos de operação, administração, manutenção e provisionamento de redes.

Realizar tais atividades de forma manual é geralmente bastante custoso, pois exige pessoal técnico altamente qualificado e também pelo fato de que a operação manual é inerentemente ineficiente. Este problema é particularmente interessante na questão da identificação e correção de falhas, pois ela é comumente relegada a um segundo plano até que comece a causar impactos mais sérios na operação da rede. No acirrado mercado de telecomunicações existente hoje, a automatização de tais processos pode representar uma vantagem competitiva significativa, na medida em que os custos de operação de redes podem ser bastante reduzidos frente à concorrência.

No que diz respeito à automatização de processos de gerência de redes, existem pelo menos quatro níveis de atividades [Stevenson95]. Estes quatro níveis de atividades representam diferentes estratégias de gerência de equipamentos e serviços a serem adotadas em uma rede, e são descritos a seguir. No nível de atividades *inativo* (i) não há nenhum tipo de monitoramento e nenhuma ação é tomada mesmo quando um alarme é recebido. Por razões óbvias, este nível só deve ser aplicado a elementos pouco relevantes para a operação de uma rede. O nível de atividades *reativo* (ii) contempla as atividades de reação a uma falha que tenha sido identificada (através de um alarme, por exemplo), mas não prevê ações de monitoramento para a identificação de falhas. No nível de atividades *interativo* (iii), é feito um monitoramento dos elementos, mas ainda é necessário buscar a causa raiz de falhas de forma manual, possivelmente interagindo com diversos elementos e identificando e descartando os efeitos colaterais da falha. Por fim, o nível de atividades *pró-ativo* (iv) prevê tanto o monitoramento de elementos de rede, quanto mecanismos automatizados de identificação de causas raízes de falhas e correção das mesmas.

Sistemas com pouca ou nenhuma pró-atividade (nível ii) são mais simples e desempenham atividades de gerência, como testes e diagnósticos de problemas, apenas sob demanda, a partir de estímulos diretos de usuários ou de outros sistemas. Por outro lado, há sistemas mais pró-ativos (níveis iii e iv) que são mais complexos e agem de forma mais autônoma, não dependendo exclusivamente de estímulos diretos de usuários. Neste caso, políticas ou metas de alto nível definidas pelo usuário são interpretadas pelo sistema que busca alcançá-las executando as atividades de gerência cabíveis. Estes sistemas representam um importante passo no sentido de tornar as redes auto-gerenciáveis, o que é um objetivo perseguido tanto pela indústria quanto pela academia.

1.3 Justificativa

O processo do crescimento da oferta de serviços de telecomunicações e do uso de sistemas específicos de gerência de redes tem ocorrido intensa e progressivamente há vários anos. Paralelamente a este processo de mercado, muitos esforços têm sido conduzidos no sentido da pesquisa e desenvolvimento de novos e mais eficientes modelos e técnicas para a gerência de redes de telecomunicações. Muitos destes esforços visam a viabilização de redes tão pró-ativas quanto possível, que possam se enquadrar nos níveis de automatização iii e iv discutidos na Seção 1.2.

Há uma conseqüência natural deste processo de evolução apresentado no parágrafo anterior. Muitos dos sistemas de gerência efetivamente utilizados em redes de produção não apresentam algumas das características e funcionalidades mais modernas desenvolvidas no contexto de gerência de redes. Obviamente, tal discrepância é mais pronunciada em redes legadas, que são baseadas em tecnologias de telecomunicação mais antigas. Geralmente por razões de ordem econômica, tais redes permanecem em operação por muitos anos mesmo depois de serem consideradas ultrapassadas.

Caracteriza-se então um interessante desafio. A permanência em operação de redes legadas é, freqüentemente, uma necessidade inegociável para algumas organizações. Por outro lado, tecnologias antigas usualmente contrastam com novos requisitos técnicos ou de mercado para os quais não foram inicialmente projetadas. Torna-se necessário, na medida do possível, adaptar-se tais redes legadas a esta nova realidade com vistas a satisfazer tais requisitos.

Naturalmente, existem algumas questões a serem consideradas e barreiras a se superar para que tal desafio possa ser superado com sucesso. Considerando-se aspectos econômicos e de negócios, uma das mais importantes questões é evitar ou reduzir ao máximo o impacto sobre as operações das operadoras de redes de telecomunicações causado pela introdução de tais processos de automatização. Este aspecto é fundamental uma vez que a principal motivação da automatização é a diminuição de custos. Outra importante questão é que os investimentos efetuados ao longo do tempo em sistemas para a gerência das redes devem ser considerados. É desejável ser capaz de se aproveitar ao máximo os recursos de gerência já existentes, como forma de se evitar maiores gastos com os novos sistemas. Tais fatores sugerem a adoção de soluções que possam ser introduzidas de forma gradativa ou ainda que sejam complementares aos sistemas já existentes.

Além dos aspectos comerciais, há também o aspecto dos desafios tecnológicos que este problema apresenta e que devem ser solucionados ou contornados. Qualquer problema que envolva redes é distribuído por natureza, o que por si

só já introduz diversas complicações. A isto se somam dificuldades advindas da heterogeneidade de tecnologias, protocolos e equipamentos existentes nas redes. Assim sendo, soluções para o problema de automatização devem ser preferencialmente distribuídas, o que lhes pode conferir maior escalabilidade frente às grandes redes em que vão atuar. Além disso, novas soluções devem ser bastante flexíveis, de forma a se adaptarem facilmente às diversas particularidades tipicamente encontradas em redes de telecomunicações.

1.4 Objetivo

Um dos novos requisitos que redes legadas devem atender é a automação de processos de gerência de redes discutida na Seção 1.2. Tendo em vista a meta de tornar tais redes auto-gerenciáveis e tão autônomas quanto possível, justifica-se a aplicação de técnicas modernas que confiram mais pró-atividade a estas redes. Entretanto, qualquer solução proposta para este problema deve levar em consideração as condições e restrições de ordem tanto comercial quanto técnica discutidas na Seção 1.3.

É neste contexto que se enquadra o presente trabalho. Seu objetivo principal é a aplicação de modernas técnicas de Engenharia de Software (principalmente agentes de software) sobre sistemas legados de gerência de redes de telecomunicações, com o fim de conferir-lhes funcionalidades extras que não foram consideradas quando de sua concepção. Os esforços deste trabalho foram concentrados particularmente na área de gerência de falhas. Contudo, também seria possível aproveitar-se algumas de suas contribuições em outras áreas como, por exemplo, gerência de configuração.

Como discutido anteriormente, qualquer solução proposta para este problema deve ser gradual e levar em conta os sistemas de gerência já existentes e as funcionalidades que estes apresentam. A solução proposta neste trabalho atende estes requisitos, pois se vale de bases de gerência previamente instaladas, utilizando dados e recursos provenientes destes sistemas. Além disso, seu funcionamento causa pouco impacto nos sistemas de gerência e na rede, pois apresenta uma arquitetura distribuída que prevê mecanismos para evitar a sobrecarga dos elementos da rede, e em particular as estações de gerência.

A solução proposta neste trabalho é baseada em uma arquitetura distribuída composta por agentes de software dispersos pela rede. Cada agente desempenha papéis específicos, cooperando uns com os outros com o fim comum de automatizar processos de detecção e diagnóstico de falhas ou más configurações de equipamentos na rede. A arquitetura proposta neste trabalho foi especificamente moldada de acordo com a realidade tipicamente encontrada em redes legadas de

telecomunicações. Tal realidade diz respeito principalmente ao tipo de sistema centralizado de gerência comumente utilizado em redes baseadas em tecnologias como ATM e Frame Relay.

Diante do problema apresentado, este trabalho propõe algumas contribuições. A primeira é a adaptação de redes legadas aos requisitos de automação de processos de gerência de redes. Tal adaptação pode ser efetivamente considerada como uma agregação de novas funcionalidades à antiga plataforma. Isto ajuda a diminuir os efeitos do distanciamento tecnológico de tal plataforma em relação a gerações mais modernas de sistemas de gerência. Outra contribuição é a estratégia utilizada para que tal agregação de funcionalidade ocorra de forma a causar o menor impacto possível sobre o sistema de gerência pré-existente. Como será discutido no decorrer do trabalho, a solução conta com uma estratégia de distribuição de informações de topologia de rede através de agentes de software que minimiza as exigências adicionais de carga sobre o sistema de gerência legado. Tal estratégia visa evitar um potencial comprometimento da performance do sistema legado. Uma outra contribuição é a modelagem de testes de forma independente dos agentes que de fato os executam sobre os elementos de rede. Esta modelagem confere uma maior flexibilidade à solução, uma vez que os artefatos que definem os testes podem ser reutilizados com mais facilidade devido à sua modularidade. Uma última contribuição é a aplicação prática de agentes de software no domínio de gerência de redes. A decisão pelo uso de agentes de software se deve ao fato de que eles representam uma abstração de alto nível bastante apropriada para o projeto de sistemas distribuídos, o que pode tornar mais facilmente administráveis os desafios impostos pelo problema de gerência de redes.

1.5 Organização

Este documento está organizado da seguinte forma. O Capítulo 2 apresenta toda a fundamentação teórica necessária para uma melhor compreensão deste trabalho. Há duas importantes seções neste capítulo. A Seção 2.1 trata de gerência de redes de forma geral. Nela são abordados os principais modelos de gerência de rede criados por organizações de padronização como a ISO, a ITU e a IETF. Há ainda nesta seção uma discussão sobre os diversos paradigmas de gerência de redes utilizados pela indústria ou sugeridos pela academia. A Seção 2.2 trata de conceitos relacionados a agentes de software. São apresentadas as principais características que diferenciam um agente de software de outras abstrações existentes na área de engenharia de software. Também são discutidos aspectos relacionados a Sistemas Multi-Agentes e possíveis arquiteturas para implementação dos mesmos. Por fim, alguns dos principais padrões na área de agentes de software são apresentados.

O Capítulo 3 apresenta a arquitetura NeMaSA. É apresentado o contexto em que esta arquitetura foi concebida assim como o problema principal que serviu de motivação para a mesma. A solução é descrita com detalhes, o que inclui a listagem dos principais tipos de agentes de software e a descrição dos mesmos. Também são abordados aspectos relacionados às interações entre os agentes da arquitetura. Por fim, são discutidas questões sobre a implementação deste projeto.

O Capítulo 4 apresenta em breve estudo de caso em que dois exemplos de testes comumente utilizados para diagnóstico de falhas em redes reais são apresentados. Neste capítulo se discute como tais testes podem ser implementados através da arquitetura proposta. O primeiro teste é bem simples, mas representa um dos tipos de testes mais importante para diagnóstico de falhas. O segundo é mais complexo e ilustra como um teste que necessita de informações de topologia de rede pode ser implementado com a arquitetura proposta.

O Capítulo 5 trata especificamente da aplicação de agentes de software na área de telecomunicações. São levantadas as principais razões e vantagens do uso de agentes de software neste domínio. São apresentados ainda diversos exemplos de trabalhos acadêmicos em que há o emprego de agentes de software para solucionar problemas em redes de telecomunicações. Destes, três trabalhos na área de gerência de redes e que apresentam características comuns à solução proposta neste trabalho são discutidos com mais detalhamento.

O Capítulo 6 encerra este trabalho com as considerações finais e indicações de futuros trabalhos no domínio de agentes de software aplicados à gerência de redes.

2
Fundamentação Teórica

Neste capítulo será feita uma discussão sobre os aspectos teóricos necessários para o embasamento do trabalho aqui apresentado. Os principais tópicos apresentados dizem respeito à gerência de redes e alguns dos mais importantes padrões de gerência criados por organizações de normatização. É através dos modelos e protocolos de gerência definidos por tais padrões que a solução proposta neste trabalho se torna efetivamente capaz de se comunicar com equipamentos de rede para identificar falhas e eventualmente corrigi-las. Também serão apresentados os principais paradigmas de gerência de redes concebidos ao longo do tempo. Em particular, a caracterização dos modelos centralizado e distribuído é bastante relevante para este trabalho. Em seguida será feita uma apresentação de aspectos teóricos relacionados a agentes de software. Explorar-se-á algumas de suas principais características, assim como a sua organização em sistemas multi-agentes.

2.1
Gerência de Redes

2.1.1
Introdução

Redes de telecomunicações são tipicamente compostas por um determinado número de nós geograficamente distribuídos, conectados entre si através de algum meio físico (como fibras ópticas, cabos coaxiais ou enlaces de rádio) que permitem o fluxo de informações entre dois ou mais pontos distintos. Estas redes permitem a comunicação entre pessoas (caso típico das redes de telefonia) ou a comunicação entre sistemas computacionais (como as redes de computadores que formam a Internet), através da transmissão de seqüências de bits (representando voz, vídeo ou dados) entre nós remotos.

Conceitualmente, uma rede de telecomunicações é composta por três planos ortogonais que contemplam diferentes aspectos da estrutura e da operação da mesma [ElSayed02]. O primeiro e principal plano é o plano de dados (ou de transporte), que é a parte da rede que transporta os dados dos usuários. Sua finalidade é bem clara uma vez que o principal objetivo de uma rede de telecomunicações é possibilitar a troca de informações entre seus usuários. Além

desta finalidade básica, funções como controle de fluxo e de erro também devem ser executadas neste plano. Para que uma rede seja capaz de funcionar corretamente, os nós devem trabalhar em conjunto para possibilitar que os dados cheguem aos seus devidos destinatários e com os níveis de serviço pré-determinados. Para tanto, informações de controle devem ser trocadas entre os nós que compõem a rede. O segundo plano, o plano de controle, ou de sinalização, é a parte da rede que lida com estas informações (ou sinais) de controle. Estes sinais permitem, por exemplo, o estabelecimento de chamadas em redes de telefonia, a criação de circuitos virtuais para a transmissão de dados em redes ATM [I.321] e a troca de informações de roteamento em redes IP [RFC791]. Há protocolos específicos de controle (ou sinalização) como o SS7 [Q.700] para telefonia, o PNNI [PNNI] para redes ATM e os protocolos BGP [RFC4271] e OSPF [RFC2328] para redes IP. O terceiro e último plano é o plano de gerência, que é a parte da rede que lida com aspectos administrativos e de manutenção dos equipamentos e serviços da rede. Funções como gerência de falhas, configuração de equipamentos, bilhetagem, desempenho e segurança são desempenhadas neste plano da rede. Alguns protocolos de gerência comumente utilizados incluem SNMP [RFC1157], CMIP [X.711] e TL1 [GR-831].

As principais tarefas tipicamente desempenhadas por sistemas de gerência são sintetizadas no termo OAM&P (*Operations, Administration, Maintenance, and Provisioning*). Operação diz respeito a atividades rotineiras de monitoramento do ambiente de rede, visando fundamentalmente à detecção, ao diagnóstico e à correção de falhas que porventura venham a ocorrer. São estas as atividades que mantêm a rede em funcionamento no curto prazo. A atividade de administração envolve fundamentalmente o planejamento da rede no longo prazo. Dados estatísticos sobre a operação e o uso da rede, tendências de mercado e estratégias organizacionais são levantados para posterior análise. Estes estudos e as decorrentes propostas de melhorias são fundamentais para que a rede permaneça confiável e adequada aos serviços que ela deve prover a seus usuários. As atividades de manutenção englobam a atualização, correção, ou substituição de equipamentos, criação e restauração de *backups*, entre outros. Normalmente são tarefas críticas que interrompem temporariamente o funcionamento da rede. Desta forma, a execução de tais tarefas deve ser planejada cuidadosamente para que causem o menor impacto possível aos usuários da rede. Por fim, as atividades de provisionamento dizem respeito à criação e ao estabelecimento de serviços de rede a um usuário. A remoção de tais serviços também é considerada uma atividade de provisionamento. Estas atividades podem envolver tanto a instalação de novos equipamentos quanto apenas a configuração de parâmetros de serviço.

Desempenhar todas essas tarefas de gerência em redes de telecomunicações é uma tarefa complexa e deve ser preferencialmente conduzida com o auxílio de sistemas de software desenvolvidos especificamente com este propósito.

De modo geral, a literatura de gerência de redes levanta três formas distintas de se gerenciar uma rede qualquer, a saber, (i) protocolos de rede (de quaisquer camadas) podem incluir neles próprios informações de gerência, (ii) protocolos de gerência específicos para uma determinada camada (de rede) podem ser estabelecidos e (iii) mecanismos de gerência podem ser implementados como uma aplicação. Estes três casos são conhecidos como operação de camada (*layer operation*), gerência de camada (*layer management*) e gerência de sistemas (*systems management*), respectivamente [Raman98]. A alternativa mais comum, e adotada pela maioria dos modelos de gerência (incluindo todos os modelos avaliados neste trabalho), é a terceira.

O fato de a gerência ser, em sua essência, uma aplicação sobre a rede implica, entre outras coisas, que os protocolos de gerência estejam na camada de aplicação. Há algumas desvantagens na adoção desta estratégia. Em primeiro lugar, ela torna mais complexo o desenvolvimento de equipamentos gerenciáveis, uma vez que estes precisam implementar protocolos de todas camadas de rede, mesmo que estes operem em camadas inferiores. Este é o caso, por exemplo, de *bridges* e de roteadores que operam na segunda e terceira camadas do modelo OSI [ISO7498], respectivamente. Uma segunda desvantagem é que, em caso de colapsos de rede com o comprometimento do funcionamento de camadas no nível de aplicação, as aplicações de gerência serão afetadas e ficarão incapazes de lidar com a crise. No entanto, apesar destas desvantagens, esta tem sido a solução preferida pela indústria e academia. Uma importante motivação para esta terceira estratégia é que aplicações de gerência geralmente são complexas, e o uso dos serviços prestados por camadas de rede intermediárias torna mais simples a implementação dos protocolos de gerência.

2.1.2
Padrões de Gerência de Redes

Introdução

Para suprir a demanda por gerência de redes de telecomunicações, os próprios fabricantes de equipamentos se encarregaram de desenvolver sistemas de gerência. Os primeiros sistemas desenvolvidos eram, entretanto, específicos para os equipamentos de um único fabricante, e seguiam modelos e protocolos de gerência proprietários. Este fato ocasionou dificuldades de interoperabilidade entre sistemas e equipamentos de diferentes fabricantes que interagiam em uma mesma

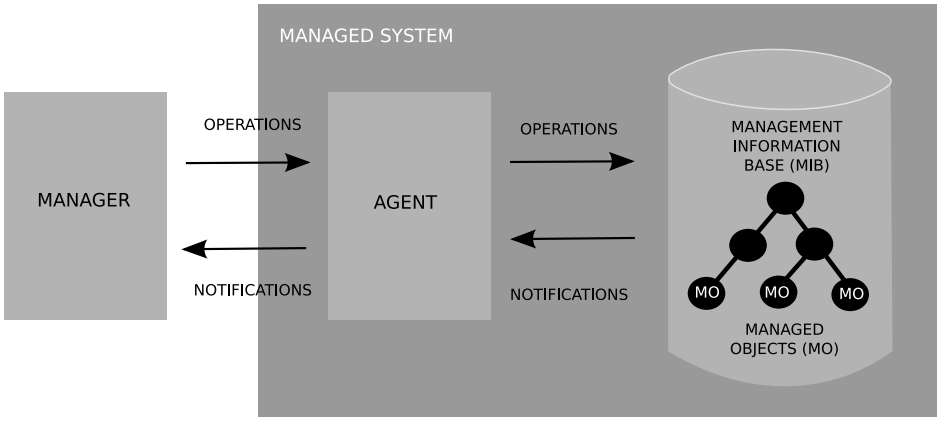

Figura 2.1: Paradigma Gerente/Agente

rede, causando grandes transtornos para os operadores da rede. Por conta disto, esforços de padronização de gerência de redes foram empreendidos por órgãos internacionais visando sanar tais questões. Estes esforços de padronização e suas principais contribuições serão abordados a seguir.

Um dos resultados mais importantes destes esforços, e que está fortemente consolidado no meio de gerência de redes, foi a arquitetura de gerência baseada no paradigma gerente/agente. De acordo com esta arquitetura, os recursos gerenciáveis de um equipamento (como por exemplo portas ou cartões) são modelados através dos chamados *objetos gerenciados*. Tais objetos gerenciados encapsulam as propriedades e funcionalidades de um recurso gerenciável e fornecem uma interface de acesso ao mesmo. Neste contexto, o agente pode ser entendido como o processo que possibilita que entidades externas acessem as propriedades e funcionalidades do conjunto de objetos gerenciados de um equipamento. O gerente, por sua vez, é a aplicação que acessa os objetos gerenciados expostos pelos agentes. O gerente tem como principal objetivo monitorar os recursos gerenciáveis e ajustá-los de acordo com os interesses dos administradores da rede. A Figura 2.1 ilustra o paradigma gerente/agente.

Cabe neste momento uma breve colocação acerca do termo “agente”. O termo “agente” tem sido amplamente utilizado pela comunidade de ciência da computação para descrever uma entidade de software qualquer que desempenha alguma tarefa de forma automática. Como esta descrição é um tanto genérica, grupos distintos desta grande comunidade acabaram atribuindo ao termo “agente” conotações distintas, trazendo, em algumas circunstâncias, dificuldades à comunicação entre os mesmos. Este conflito de nomenclatura é particularmente flagrante entre o grupo de gerência de redes e os grupos de inteligência artificial e de engenharia de software. Na literatura de gerência de redes, o termo agente é freqüentemente

usado para descrever o software embutido em equipamentos gerenciáveis (tais como roteadores ou *switches*) que responde a requisições feitas por estações de gerência e que envia alarmes à mesma [Yemini93, Stallings98a]. Este modelo de "agente" discutido no parágrafo anterior segue um paradigma cliente-servidor e é utilizado pelos mais importantes padrões de gerência de redes em uso na indústria apresentados a seguir. Por outro lado, a literatura de inteligência artificial e, mais recentemente, a de engenharia de software aplicam o termo "agente" para designar programas independentes capazes de agir de forma autônoma que buscam atingir metas pré-estabelecidas [Wooldridge95]. Este trabalho emprega o termo "agente" de acordo (primariamente) com a perspectiva da comunidade de engenharia de software. Entretanto, na presente seção (em que se abordam os padrões mais importantes de gerência de redes) este termo será freqüentemente empregado de acordo com a perspectiva da comunidade de gerência de redes. Espera-se que o contexto em que o termo estiver presente seja suficiente para dirimir eventuais ambigüidades.

Padrões de Gerência de Redes

Há algumas organizações que, ao longo dos anos, desenvolveram serviços, protocolos e arquiteturas para gerência de redes. Algumas delas são responsáveis pelos mais importantes padrões de gerência de redes utilizados pelo mercado. Três das mais importantes são citadas a seguir [Pras95]:

- International Organization for Standardization (ISO);
- International Telecommunication Union (ITU);
- Internet Engineering Task Force (IETF).

A ISO e a ITU, por exemplo, são autoras do importante modelo de referência OSI [ISO7498] de sete camadas, que engloba algumas das mais importantes abstrações usadas no ensino de conceitos de rede. Outras importantes contribuições destas organizações foram a pilha de protocolos que implementa o modelo de referência, assim como o protocolo de gerência CMIP [X.711].

Uma das mais importantes contribuições do ITU nesta área foi o TMN [M.3010], que é um modelo de gerência aplicável a redes de telecomunicações. Este modelo incorpora e usa muitos dos conceitos estabelecidos pelo modelo OSI e, embora não seja comumente aplicado em redes de dados IP (baseadas em comutação de pacotes), é utilizado freqüentemente para gerência de redes SDH [G.707] e GSM [Rahnema93], por exemplo [Tanaka98, Towle95].

A IETF, por sua vez, é responsável pela criação de muitos dos padrões da Internet, dentre os quais se encontram a pilha de protocolos TCP/IP [RFC1180]

e, em particular, o protocolo SNMP [RFC1157] utilizado para gerência de equipamentos de rede. Em alguns aspectos, os padrões da IETF se sobrepõem a padrões da ISO e da ITU. Esta sobreposição é bem nítida, por exemplo, quando se compara a pilha de protocolos TCP/IP (IP, OSPF, TCP e SNMP, entre outros) com a pilha de protocolos OSI (CNLP, IS-IS, TP4 e CMIP, entre outros). Detalhes sobre estas duas pilhas de protocolos podem ser encontrados em [Cisco03]. As principais razões que levaram a IETF e a ISO/ITU a seguir estas diferentes direções foram a demora para a conclusão dos padrões e protocolos OSI e, posteriormente, a relativa complexidade dos mesmos. Estes fatores levaram a IETF a desenvolver padrões próprios mais simples que os da OSI/ITU [Pras95].

A seguir, alguns aspectos importantes dos modelos e protocolos de gerência criados por essas organizações serão abordados.

Modelo de Gerência OSI

Por volta do início da década de 1980, soluções de redes de dados comumente utilizadas (como a SNA da IBM [Cisco03] e a DECnet da DEC [Cisco03]) eram baseadas em tecnologias proprietárias que não se integravam facilmente umas com as outras. Esta característica muitas vezes impactava negativamente a capacidade de usuários resolverem seus problemas através da integração de suas redes e sistemas. Como resposta a esta importante deficiência, a ISO (juntamente com a ITU) desenvolveu o modelo de uma rede padronizada e não-proprietária. Tal modelo, conhecido como modelo OSI, tinha como objetivo eliminar os problemas da pouca interoperabilidade então vivenciada e, desta forma, trazer mais flexibilidade a seus usuários [Russell06].

Algumas das contribuições mais importantes deste trabalho da ISO e ITU foram a criação do modelo de redes de sete camadas, a especificação de protocolos pertencentes a cada uma destas camadas, e, particularmente, o desenvolvimento do modelo de gerência OSI que será discutido a seguir.

O modelo de gerência OSI provê três importantes contribuições para a gerência de redes. Estas contribuições são tanto práticas quanto teóricas e representam um marco importante nesta área. Estas três contribuições são (i) o conceito das áreas funcionais, (ii) modelos de informação para representar recursos de rede e (iii) protocolos para transferência de informações sobre gerência de redes [Raman98]. Estes três aspectos do modelo de gerência serão abordados a seguir com mais detalhes.

A primeira contribuição do modelo de gerência OSI diz respeito ao conceito das áreas funcionais. Cada área funcional representa um tipo de atividade que a gerência de uma rede de telecomunicações deve desempenhar. As cinco áreas

levantadas pela OSI são também conhecidas pelo termo "FCAPS", formado a partir das iniciais de cada área (em inglês) e são sucintamente apresentadas a seguir.

Gerência de falhas. Responsável pela detecção, isolamento, notificação de administradores e, na medida do possível, correção de falhas na rede;

Gerência de configuração. Responsável pelo registro e manutenção dos parâmetros de configuração dos elementos de rede, assim como informações sobre versões de hardware e de software de cada um deles;

Gerência de contabilidade. Responsável pelo registro do uso da rede por parte de seus usuários com objetivo de posterior cobrança ou regulação de uso da mesma;

Gerência de desempenho. Responsável pela medição e disponibilização de informações sobre aspectos de desempenho da rede. Estes dados são usados para garantir que a rede opere em conformidade com os níveis de qualidade de serviço acordados com seus usuários;

Gerência de segurança. Responsável por restringir o acesso à rede por parte de elementos não autorizados e impedir que seus usuários a utilizem de forma irregular, intencionalmente ou não.

A segunda grande contribuição do modelo de gerência OSI foram os modelos para representação dos recursos de rede e de informações relacionadas a estes recursos. Para tal modelagem foi adotado o paradigma de orientação a objetos. Segundo esta modelagem, cada *objeto gerenciado* representa uma visão de um determinado recurso da rede que pode ser coordenada por um sistema qualquer de gerência. Exemplos de recursos de rede que geralmente são representados como objetos gerenciados são: portas de linha, cartões (com ou sem portas) e nós (equipamentos como roteadores ou *switches*). O fato de que objetos gerenciados representam visões de seus respectivos recursos significa que apenas alguns aspectos destes recursos são expostos a sistemas de gerência.

Como é de se esperar, por se tratar de um modelo orientado a objetos, existe o conceito de *classes* de objetos gerenciados, assim como o de *instâncias* de objetos gerenciados. Estes conceitos são equivalentes aos conceitos de classes e instâncias de objetos, respectivamente, do paradigma de orientação a objetos. Além disso, cada objeto gerenciado possui *atributos* e *ações*, equivalentes a atributos e métodos, respectivamente, em orientação a objetos. Estes atributos e ações podem ser agrupados em *pacotes* para facilitar a reutilização de especificações. Por fim, cada objeto gerenciado pode estar associado a *comportamentos*, que descrevem de

forma geral o modo de atuação do objeto, assim como *notificações*, que especificam os alarmes que o objeto pode emitir.

A especificação destes objetos é feita através das linguagens de notação ASN.1 [X.680], que descreve a sintaxe das estruturas de dados usadas pelos objetos, e GDMO [X.722], que descreve a estrutura dos objetos gerenciados propriamente ditos.

A identificação de um objeto gerenciado em uma base de objetos correspondente a uma rede de telecomunicações deve ser feita de modo inequívoco e sem ambigüidades. Para tanto os objetos são organizados de modo hierárquico em uma estrutura em forma de árvore. Isto é possível, uma vez que objetos gerenciados podem conter referências a outros objetos gerenciados. Há, entretanto, a restrição de que o valor do atributo de identificação de cada objeto contido por um mesmo objeto pai seja único. Esta restrição e a organização em árvore permitem a identificação inequívoca de um objeto gerenciado na chamada *Management Information Tree* (MIT).

A terceira importante contribuição do modelo OSI diz respeito aos protocolos e serviços que permitem a troca de informações entre equipamentos de rede e sistemas de gerência. Os serviços de gerência providos aos usuários são especificados pelo *Common Management Information Service* (CMIS) [X.710]. O protocolo que implementa estes serviços é conhecido como *Common Management Information Protocol* (CMIP) [X.711]. Apesar de haver uma clara separação entre serviço e protocolo, o termo CMIP é comumente utilizado para referir-se a ambos.

A Tabela 2.1 indica os diversos serviços oferecidos pelo CMIP. Estes serviços podem ser divididos em duas classes distintas. A primeira engloba reportes enviados autonomamente peloos elementos gerenciados aos sistemas de gerência para avisá-los sobre eventos relevantes. Apenas o serviço M-EVENT-REPORT se enquadra nesta classe. A segunda classe engloba as requisições feitas aos elementos gerenciados oriundas dos sistemas de gerência e as respectivas respostas. Todos os demais serviços se enquadram nesta classe.

Através destes serviços é possível ter acesso aos objetos gerenciados, tanto aos seus atributos quanto aos seus métodos (ações), bastando-se para tal, conhecer a posição do objeto na MIT. Há ainda dois mecanismos que permitem ao usuário do CMIP um maior controle sobre o universo de objetos envolvidos em uma determinada operação de gerência. Estes são os mecanismos de escopo (*scoping*) e filtragem (*filtering*) e estão disponíveis para os serviços M-GET, M-SET, M-ACTION e M-DELETE [Stallings93]. O mecanismo de escopo permite restringir o universo sobre o qual uma consulta ou ação referente a um serviço será aplicado. Esta restrição é feita determinando-se uma sub-árvore da MIT e a quantidade de níveis nesta sub-árvore sujeitos à ação do serviço. O mecanismo

Tabela 2.1: Serviços oferecidos pelo CMIP

Serviço	Descrição
M-EVENT-REPORT	Reporta um evento em um elemento gerenciado a um sistema de gerência
M-CREATE	Requisita a criação de uma instância de um objeto gerenciado
M-DELETE	Requisita a destruição de uma instância de um objeto gerenciado
M-GET	Requisita a busca de informação em um objeto gerenciado
M-SET	Requisita a modificação de informação em um objeto gerenciado
M-ACTION	Requisita que um elemento gerenciado execute uma ação
M-CANCEL-GET	Requisita que uma solicitação M-GET pendente seja cancelada

de filtragem também restringe o universo de objetos gerenciados sobre o qual um serviço CMIP será aplicado. Ele se dá através de um conjunto de expressões booleanas com assertivas sobre a presença ou o valor de um ou mais atributos dos objetos gerenciados. Apenas os objetos que satisfaçam estas expressões booleanas estarão sujeitos à operação de gerência em questão. Este mecanismo é semelhante, apesar de ser menos flexível, a uma cláusula WHERE em uma consulta SQL.

Modelo de Gerência TMN

Desde a década de 1980, a ITU tem trabalhado na especificação de um modelo genérico de gerência de redes de telecomunicações. A este modelo deu-se o nome *Telecommunications Management Network* (TMN) [M.3010]. Um dos propósitos deste modelo é ser aplicável em qualquer rede de telecomunicações, não importando as tecnologias de rede utilizadas ou o grau de heterogeneidade de fornecedores de equipamentos existente na mesma. Para tanto, foram definidos modelos e recomendações que permitissem, de forma padronizada, a operação, administração, manutenção e provisionamento (OAM&P) de redes e seus serviços nos complexos e heterogêneos ambientes tipicamente encontrados nas operadoras de telecomunicações.

O modelo TMN foi fortemente influenciado pelo modelo de gerência OSI, adotando alguns conceitos criados ou escolhidos por este. De fato, os dois modelos foram criados conjuntamente pela ISO e a ITU. Um exemplo da influência do modelo de gerência OSI sobre o padrão TMN foi a adoção do modelo gerente/agente e do paradigma de orientação a objetos.

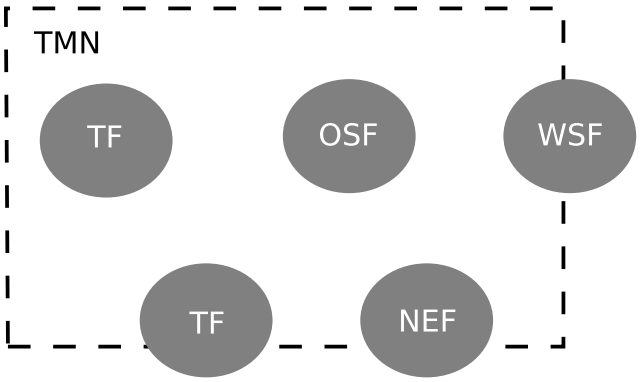

Figura 2.2: Blocos de função do TMN

Tabela 2.2: Blocos de função do TMN

Sigla	Nome	Descrição
OSF	Operation System Function block	Desempenha tarefas como monitoramento, coordenação e controle
NEF	Network Element Function block	Bloco de função correspondente a um elemento de rede gerenciado
WSF	Work Station Function block	Permite a interação de seres humanos com a TMN
TF	Transformation Function block	Conecta entidades funcionais com mecanismos de comunicação incompatíveis

De acordo com a Recomendação M.3010 da ITU-T, a TMN é, conceitualmente, uma rede de sistemas de gerência separada da rede de telecomunicações, mas que possui interfaces com os elementos de rede em pontos específicos. A arquitetura TMN definida nesta recomendação apresenta as entidades de uma rede TMN, assim como as comunicações entre estas, através de três diferentes perspectivas, a saber:

- Arquitetura funcional;
- Arquitetura física;
- Arquitetura de informação.

A arquitetura funcional define as principais funcionalidades sujeitas a padronização. Estas funcionalidades são mapeadas nos nós de uma TMN quando da implementação da mesma. Cada uma destas funcionalidades pode ser considerada uma entidade lógica da rede e recebe o nome de *bloco de função*. Há quatro tipos principais de blocos de função definidos no TMN, exibidos na Figura 2.2 e descritos na Tabela 2.2.

O bloco de função OSF é quem de fato desempenha as tarefas de gerência da rede. Ele monitora, coordenada e controla as funções de gerência da rede.

Tabela 2.3: Pontos de referência do TMN

Nome	Descrição
q	Relaciona OSFs a NEFs, a TFs e a outros OSFs. Também relaciona TFs a NEFs e a outros TFs.
x	Relaciona duas OSFs em TMNs diferentes.
f	Relaciona WSFs a OSFs e a TFs.
g	Relaciona WSFs a usuários humanos.
m	Relaciona TFs a equipamentos não-TMN.

Blocos OSFs podem estar conectados entre si, formando estruturas hierárquicas, por exemplo, para melhor desempenhar suas atividades. Também é possível conectar OSFs pertencentes a diferentes TMNs.

O bloco de função NEF apresenta a funcionalidade de um equipamento de telecomunicações, que é conduzir dados entre dois usuários da rede. Além disto, por ser o alvo principal de todas as atividades de gerência, este bloco deve possuir algum tipo de funcionalidade que permita que outros blocos de função, em particular o OSF, o gerenciem.

O bloco de função WSF provê os meios para que as informações de gerência sejam acessíveis aos administradores de redes, possibilitando que estes atuem manualmente na gerência da rede.

O bloco de função TF tem como finalidade possibilitar a conexão de duas entidades funcionais que apresentem mecanismos de comunicação incompatíveis entre si. Desta forma, ele age como uma espécie de *gateway* entre ambos. O bloco de função TF pode estar presente tanto no interior de uma TMN ligando dois outros blocos de função (padronizados mas com interfaces incompatíveis), quanto na fronteira de uma TMN servindo como canal de comunicação entre duas TMNs diferentes ou entre uma TMN e um ambiente não-TMN.

Os blocos de função interagem entre si através dos chamados *pontos de referência*. Os pontos de referência representam uma visão externa dos serviços oferecidos por um bloco de função. As ligações (ou interações) lógicas entre os blocos de função se dão através dos pontos de referência.

O TMN define cinco classes diferentes de pontos de referência, a saber: *q*, *x*, *f*, *g* e *m*. Estes pontos de referência permitem o relacionamento entre blocos de função conforme pode ser observado na Tabela 2.3. As classes q, x e f são descritas com algum detalhe na especificação. As classes g e m, entretanto, relacionam entidades externas à TMN e são descritas de forma parcial. A Figura 2.3 ilustra os blocos de função e os pontos de referência da arquitetura funcional do TMN.

Além da arquitetura funcional, o TMN também define uma arquitetura física. O propósito da arquitetura física é definir o mapeamento dos elementos da arquitetura funcional nos equipamentos físicos. A arquitetura física é composta

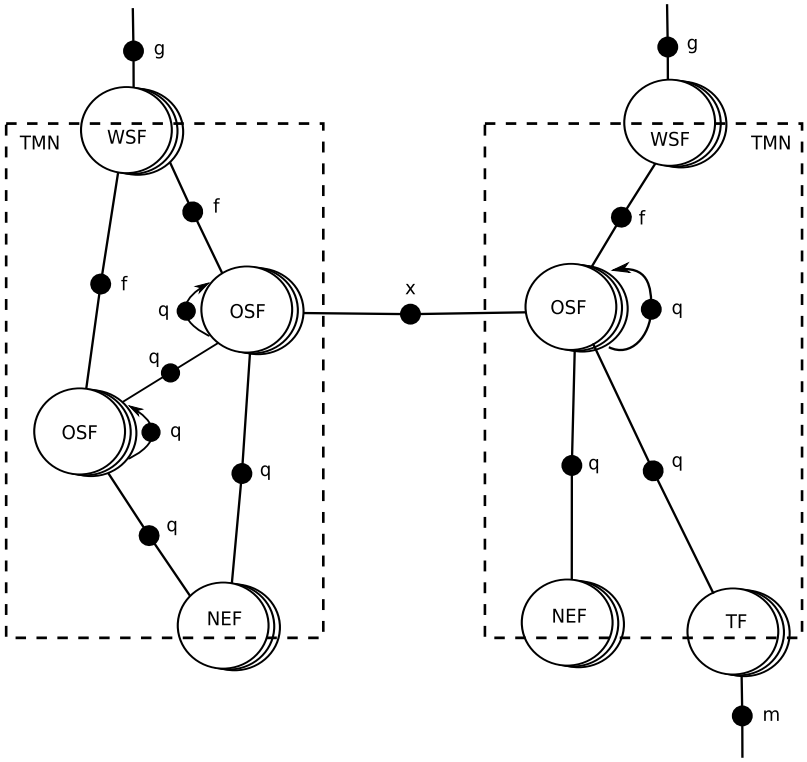

Figura 2.3: Arquitetura funcional TMN

de *blocos físicos* e *interfaces físicas*. Os blocos físicos representam os sistemas e equipamentos reais de uma rede de gerência que assumem a funcionalidade de um bloco de função da arquitetura funcional. Um bloco físico pode assumir a funcionalidade de mais de um bloco funcional. Os possíveis mapeamentos entre blocos de função e blocos físicos podem ser vistos na Tabela 2.4.

Como pode ser observado na Tabela 2.4, o padrão considera dois tipos diferentes de transformação, a saber, a adaptação e a mediação, que se aplicam aos pontos de referência q e x. Por esta razão há quatro blocos físicos relacionados ao bloco de função TF. A adaptação possibilita a comunicação entre um bloco físico TMN e uma entidade física não-TMN. A medição, por sua vez, permite a comunicação entre blocos físicos TMN que possuam mecanismos de comunicação incompatíveis entre si.

Há também um mapeamento direto entre pontos de referência e interfaces físicas. Este mapeamento está exposto na Tabela 2.5.

Como pode ser visto na Tabela 2.5, não há interfaces físicas definidas para os pontos de referência g e m. Isto se deve ao fato de que estes pontos de referência relacionam entidades externas ao TMN e, por conseqüência, a definição das interfaces associadas a estes pontos fica fora do escopo do padrão.

Tabela 2.4: Mapeamento entre blocos físicos e blocos de função

	NEF	TF	OSF	WSF
Network Element (NE)	M	O	O	O
Q-Adapter (QA), X-Adapter (XA), Q-Mediation (QM) e X-Mediation (XM)		M		
Operations System (OS)		O	M	O
Workstation (WS)				M

M: mapeamento mandatório; O: mapeamento opcional

Tabela 2.5: Mapeamento entre interfaces e pontos de referência

Interface física	Ponto de referência
Q	q
X	x
F	f

Apesar de o padrão TMN definir entidades como interfaces físicas, blocos físicos, pontos de referência e blocos de função, o grupo de entidades em que se concentraram os maiores esforços de padronização foi o das interfaces físicas. Pontos de referência e blocos de função são apenas abstrações e não há a necessidade de serem padronizados. Por outro lado, padronizar os blocos físicos poderia limitar ou dificultar desnecessariamente a implementação dos mesmos. Além do mais, a definição das interfaces físicas é suficiente para garantir a interoperabilidade entre os blocos físicos [Glitho95].

De todas as interfaces físicas, a que foi melhor definida foi a Q. Em versões anteriores do padrão, esta interface era denominada Q3. Esta antiga denominação ainda é bastante popular. De forma geral, o padrão define para as interfaces físicas (i) os protocolos de comunicação que podem ou devem ser usados, dentre os quais o mais relevante é o CMIP, (ii) diretrizes e recomendações relacionadas à documentação destas interfaces e (iii) diretrizes e recomendações para definição de novas classes de objetos gerenciados.

A terceira arquitetura definida pelo TMN é a arquitetura de informação. A TMN adota o modelo de representação baseado em objetos gerenciados e o modelo de comunicação gerente/agente usados pelo modelo de gerência OSI. Uma importante contribuição do padrão TMN neste terreno foi a definição de um conjunto de objetos gerenciados básicos usados para representar recursos de rede comuns (como equipamentos e circuitos) e também para desempenhar tarefas comuns (como envio de alarmes). Esta definição pode ser encontrada principalmente na Recomendação M.3100 [M.3100].

Por fim, uma última importante contribuição do modelo de gerência TMN foi a divisão das responsabilidades de sistemas de gerência de redes em quatro níveis

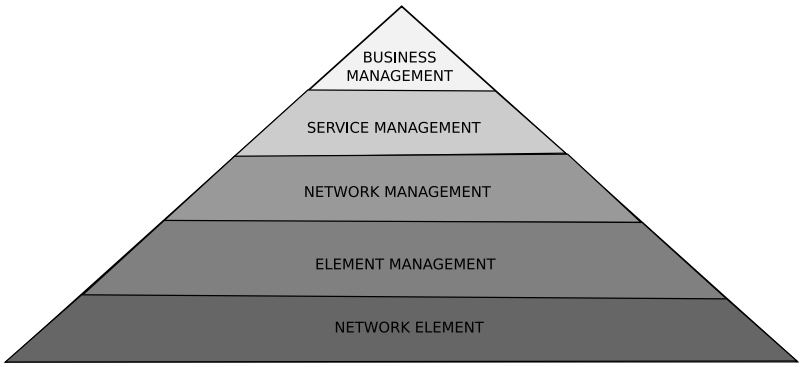

Figura 2.4: Camadas de gerência TMN

hierárquicos. Estes níveis são conhecidos como camadas de gerência e estão listados a seguir:

Camada de gerência de negócio. Esta camada tem um escopo bem amplo e engloba a gerência da organização como um todo. Um sistema de gerência pertencente a esta camada pode ser usado, por exemplo, como um suporte a processos de tomada de decisão relacionados a investimentos em recursos de rede. Dois outros possíveis usos seriam a gerência orçamentária e a gerência de recursos humanos relacionadas à gerência da rede;

Camada de gerência de serviço. Sistemas desta camada gerenciam aspectos relacionados aos serviços prestados aos clientes da organização, tais como configurações e provisionamento de serviços de comunicação. Um ponto importante monitorado nesta camada são os indicadores de qualidade de serviço (QoS);

Camada de gerência de rede. Sistemas desta camada gerenciam aspectos diretamente relacionados à interação entre equipamentos de rede, como por exemplo roteamento e congestionamento de enlaces;

Camada de gerência de elemento de rede. Sistemas desta camada gerenciam aspectos diretamente ligados a um equipamento de rede, tais como, coleta de estatísticas e detecção de erros.

A Figura 2.4 ilustra as camadas de gerência definidas pelo TMN. Há ainda uma quinta camada não gerencial que engloba os equipamentos de rede propriamente ditos.

Modelo de Gerência da Internet

O modelo de gerência da Internet como se conhece hoje surgiu a partir do final da década de 1980 em função da necessidade de se gerenciar redes IP, uma vez que estas já vinham apresentando considerável crescimento à época, o que dificultava seu gerenciamento. Neste período, os trabalhos de padronização de gerência de redes da ISO e da ITU ainda estavam longe de serem concluídos e não podiam ser aplicados em redes reais. Por tal razão, a IETF se empenhou em construir um modelo alternativo mais simples que pudesse ser concluído e implementado rapidamente. Na época, imaginou-se que este novo modelo seria apenas uma solução provisória para o problema de gerência de redes IP. O planejamento era que o modelo de gerência da Internet (baseado no protocolo SNMP) fosse eventualmente abandonado e substituído pelo modelo de gerência OSI (usando o protocolo CMIP) quando este fosse concluído [RFC1052]. Atualmente observa-se que isto acabou não ocorrendo: o modelo de gerência da Internet se tornou muito popular e o modelo OSI sobrevive apenas em nichos isolados.

O modelo de gerência de redes aplicado a redes IP possui quatro elementos principais [Stallings98a], a saber:

- Estação de gerência;
- Agente de gerência;
- Base de informações de gerência (MIB);
- Protocolo de gerência de redes.

A estação de gerência é o dispositivo em que se encontram as aplicações de gerência que efetuam tarefas tais como recuperação de falhas, configuração de equipamentos e análise de dados da operação da rede. É também através da estação de gerência que os operadores da rede podem monitorar e controlar a mesma. Para tanto, cada estação de gerência deve ser capaz de trocar informações de gerência com os equipamentos gerenciados.

O agente de gerência é o software presente nos equipamentos de rede que permite que estes sejam gerenciados remotamente. Este agente deve responder a requisições provenientes de estações de gerência solicitando informações, assim como enviar de forma pró-ativa alarmes a estações de gerência em caso de problemas.

As informações sobre os recursos da rede e o seu funcionamento devem ser estruturadas de forma padronizada para que os sistemas de gerência possam consultá-los e agir sobre os mesmos. Com este propósito foi definido o conceito da MIB, que nada mais é do que um conjunto de variáveis e tabelas que descrevem diversos aspectos do equipamento gerenciado. As MIBs são mantidas pelo agente

de gerência e cada equipamento de rede pode possuir uma ou mais MIBs de acordo com as funcionalidades e os protocolos que este implementa. Há diversas MIBs padronizadas pela IETF (assim como outras organizações) que consolidam as características mais comuns dos equipamentos de rede. Além disso, também é permitida a criação de MIBs proprietárias para a gerência das funcionalidades específicas de equipamentos que não sejam contempladas por MIBs padronizadas.

Por fim, existe o protocolo de gerência que permite que as informações de gerência sejam transmitidas entre agentes e gerentes. Este protocolo é o SNMP (*Simple Network Management Protocol*) [RFC1157], que é o elemento mais popular deste modelo de gerência. O SNMP apresenta três principais serviços: (i) *GET*, que possibilita a gerentes a obtenção de informações de agentes; (ii) *SET*, que permite que gerentes modifiquem parâmetros de configuração em agentes, e (iii) *TRAP*, que permite que agentes notifiquem sistemas de gerência acerca de problemas ou outros eventos relevantes. O serviço SET também pode ser usado para que um agente execute alguma ação sobre seu equipamento gerenciado. Esta ação normalmente é um efeito direto da mudança do valor de uma variável. Por exemplo, mudar o valor do status operacional de uma porta pode habilitá-la ou desabilitá-la. Tal comportamento deve ser documentado na especificação da MIB em que esta variável é definida.

Uma característica importante do protocolo SNMP é que ele normalmente opera acima do protocolo UDP [RFC768]. Isto faz com que as comunicações entre gerentes e agentes não sejam plenamente confiáveis, uma vez que os pacotes podem se perder ou chegar fora de ordem. As aplicações de gerência devem levar em conta esta característica e desenvolver seus próprios mecanismos para contorná-la. Um desafio particularmente difícil, entretanto, é garantir a confiabilidade no envio de TRAPs, uma vez que estes não são confirmados (o agente nunca recebe uma resposta indicando que seu alarme foi recebido com sucesso) e o sistema de gerência a princípio não tem ciência de que estes são enviados.

O protocolo SNMP é um componente central do modelo de gerência da Internet, e, ao longo de sua trajetória, três versões principais do mesmo foram definidas. As novas versões foram criadas basicamente para contornar deficiências da primeira versão. Segundo [Pras04], entretanto, é bastante improvável que haja uma nova versão do SNMP.

A primeira versão do SNMP (também conhecida como SNMPv1), definida em 1990 [RFC1157], foi fortemente inspirada nos conceitos de gerência de rede existentes na OSI antes da abordagem orientada a objetos ter sido escolhida. A idéia dos serviços GET e SET, o uso do paradigma gerente/agente e o uso da linguagem ASN.1, por exemplo, são comuns a ambos os modelos. Esta primeira versão do protocolo define cinco tipos de pacotes, a saber: (i) *GetRequest* e

(ii) *GetNextRequest*, para o serviço GET, (iii) *SetRequest*, para o serviço SET, (iv) *Response*, para respostas aos serviços GET e SET, e (v) *Trap*, para o serviço TRAP.

Um dos principais objetivos dos autores do SNMP foi criar um protocolo de gerência simples. Esta simplicidade acabou sendo um dos mais importantes fatores que contribuíram para a popularidade alcançada pelo SNMP. Por conta disto, entretanto, algumas funcionalidades foram deixadas de lado. Os três pontos mais criticados do SNMPv1 [Stallings98a] são a ineficiência no transporte de grandes blocos de dados, a inexistência de mecanismos de gerência descentralizada e a falta de segurança inerente ao protocolo.

A segunda versão do SNMP (SNMPv2), de 1993 [RFC1441], foi concebida com o intuito de solucionar os problemas listados acima. A ineficiência no transporte de grandes blocos de dados se devia basicamente a limitações dos pacotes GetRequest e GetNextRequest. Uma das limitações era que a transmissão de uma tabela (de uma MIB) devia ser feita linha a linha. Isto exigia diversas transações entre o gerente e o agente. Para resolver esta questão foi criado o pacote *GetBulk*, que permite a transmissão de mais de uma linha de uma tabela, reduzindo a quantidade de transações necessárias para a transmissão de tabelas.

Outra limitação da primeira versão do SNMP era que as transações GET eram atômicas. Isto significa que, por exemplo, quando uma solicitação GetRequest indica dois ou mais dados a serem buscados em uma MIB, e a busca de pelo menos um deles apresenta falha, nenhum dado é transmitido ao gerente. Ou o agente envia o conjunto completo de dados solicitados ou ele não envia dado algum, sinalizando que houve erro. Nestas situações de erro, o gerente deve diminuir o conjunto de dados sendo solicitados e efetuar uma nova transação. A melhoria da segunda versão em relação a esta questão foi o relaxamento da restrição de atomicidade das requisições GET. Com este relaxamento, todos os dados que puderem ser obtidos sem falhas serão enviados ao gerente, diminuindo a necessidade de novas transações.

Uma das críticas feitas ao SNMPv1 diz respeito à ausência de mecanismos que suportem uma arquitetura distribuída de gerência. O principal problema do modelo centralizado a que o SNMPv1 induz é que ele não escala bem na medida em que o número de agentes a ser gerenciado por um mesmo gerente aumenta. Em função disto, o SNMPv2 introduz um conceito de gerência distribuída, em que dois ou mais gerentes podem se organizar e dividir entre si a responsabilidade de controlar um grande conjunto de agentes. Duas entidades novas foram criadas para possibilitar a cooperação entre gerentes. A primeira é a MIB Gerente-Gerente [RFC1451], através da qual a cooperação entre gerentes pode ser conduzida. A comunicação entre os gerentes é feita através dos mecanismos comuns de GETs e

SETs, o que significa que um dos gerentes deve assumir o papel de agente. Outra nova entidade foi o pacote *Inform*, que permite que informações não solicitadas sejam enviadas de um gerente a outro, o que possibilita a notificação de eventos relevantes.

A especificação do SNMPv2 publicada em 1993 contemplava também mecanismos de segurança para transações SNMP. Esta proposta de segurança, entretanto, não foi muito bem recebida pela comunidade em função de uma falta de consenso com relação às estratégias adotadas e em função de ainda possuir deficiências [Stallings98b]. Por isto, foi publicada em 1996 uma nova revisão do SNMPv2 sem os aspectos de segurança. Esta versão revisada ficou conhecida como SNMPv2c.

Com as questões de segurança do SNMP ainda não resolvidas, novos trabalhos foram conduzidos e acabaram resultando na versão três do SNMP (SNMPv3) [RFC3411] primeiramente introduzida em 1998. O SNMPv3 inclui três importantes novos serviços, a saber, autenticação, privacidade e controle de acesso, que são brevemente discutidos a seguir.

O mecanismo de autenticação do SNMPv3 garante que uma mensagem recebida tenha sido de fato transmitida pela entidade cuja identificação consta no cabeçalho da mensagem. Também há a garantia de que esta mensagem não tenha sido alterada em trânsito nem artificialmente atrasada ou retransmitida. Este mecanismo de autenticação é baseado em chaves secretas previamente configuradas nas duas entidades (agente e gerente) que desejam se comunicar.

Para garantir a privacidade na troca de mensagens entre agentes e gerentes, é utilizado um mecanismo de criptografia baseado no DES [Menezes96]. Também é necessário que uma chave secreta seja previamente configurada em ambas as entidades. Este mecanismo de privacidade garante que nenhum terceiro seja capaz de decifrar mensagens SNMP eventualmente interceptadas.

O terceiro serviço de segurança introduzido nesta última versão do SNMP foi o controle de acesso. Com este serviço, um agente SNMP pode definir diferentes níveis de acesso à sua MIB, restringindo o acesso de gerentes à mesma. Há dois tipos principais de operações de limitação de acesso que podem ser impostas a gerentes. Com a primeira o agente pode limitar a parte da MIB visível a um determinado conjunto de gerentes. A segunda permite que o agente restrinja as operações SNMP que determinado conjunto de gerentes pode efetuar sobre sua MIB. Esta segunda limitação pode ser usada, por exemplo, para permitir apenas leituras na MIB (para tanto apenas o serviço GET seria liberado). Para que este serviço seja aproveitado, as políticas de segurança de cada gerente devem ser previamente configuradas no agente.

Outros Modelos (ou Protocolos) de Gerência de Redes

Os modelos e protocolos de gerência de redes apresentados nas seções anteriores podem ser considerados como os mais importantes em sua categoria. Isto se deve ao grau de influência que eles têm exercido na evolução da área de gerência de redes e também na grande aceitação e aplicação dos mesmos pela indústria nas redes que oferecem os serviços de telecomunicações utilizados pelas sociedades modernas. Entretanto, outros modelos ou padrões de gerência têm surgido ao longo do tempo e contribuído para o desenvolvimento de redes mais facilmente gerenciáveis. Alguns destes são citados a seguir.

O TINA-C (*Telecommunications Information Networking Architecture Consortium*) [Barr93] é um consórcio formado por empresas de telecomunicações no início da década de 1990. Ele tem como objetivo prover uma arquitetura baseada em tecnologias distribuídas (como CORBA [OMG04]) que possibilite a operadores de redes de telecomunicações a introdução rápida e flexível de novos serviços, assim como a gerência de serviços e da infra-estrutura de rede de forma integrada.

O RMON (*Remote Monitoring*) [RFC3577] é uma especificação de monitoramento que permite que *probes* localizadas em uma rede coletem dados estatísticos sobre a operação da mesma. Estes dados coletados são consolidados em uma MIB padronizada e podem ser transmitidos via SNMP a uma estação de gerência.

O WBEM (*Web-Based Enterprise Management*) [WBEM] é um conjunto de modernas tecnologias de gerência de sistemas desenvolvido para ambientes de rede, em especial redes de computadores corporativas. Estes padrões são desenvolvidos pela organização DMTF (*Distributed Management Task Force*). Fazem parte do WBEM tecnologias tais como o CIM (*Common Information Model*) [CIM], que é um modelo para descrever entidades comuns a sistemas de informação, e o WS-Management (*Web Services for Management*) [WSMAN], que permite a gerência de redes e aplicações através de Web Services [WS]. As tecnologias do WBEM têm sido adotadas por sistemas operacionais de diversos fornecedores. Exemplos desta adoção são o WMI (*Windows Management Instrumentation*) [Tunstall02] da Microsoft (Windows), o WBEM Services [WBEMServices] da Sun (Solaris) e o OpenWBEM [OpenWBEM] da Novell (SUSE Linux).

2.1.3 Paradigmas de Gerência

Com o avanço das tecnologias de redes de telecomunicações e das técnicas de gerência das mesmas, diferentes modelos e paradigmas de gerência de redes têm sido propostos e implementados. Um dos aspectos comuns aos modelos mais importantes apresentados na Seção 2.1.2 é a arquitetura baseada no paradigma

gerente/agente criada pela ISO e ITU e popularizada pelo SNMP. Em alguns casos, utilizam-se variações sofisticadas desta arquitetura que empregam vários gerentes organizados em diferentes níveis de gerência, responsáveis por tarefas específicas e que formam uma estrutura hierárquica de gerência de redes.

Não há no meio acadêmico um consenso universal sobre a classificação (ou mesmo sobre questões de nomenclatura) de modelos de sistemas de gerência, mas três modelos são comumente citados pela maioria dos pesquisadores, a saber: o centralizado, o hierárquico e o distribuído. Os modelos hierárquico e distribuído também são conhecidos como fracamente distribuído e fortemente distribuído, respectivamente. Um estudo mais completo sobre esta questão pode ser encontrado em [MartinFlatin99]. Neste trabalho são apresentadas duas taxonomias para classificação de sistemas de gerência de rede. A mais simples estabelece quatro principais paradigmas utilizados por sistemas de gerência de redes, a saber:

- Paradigma centralizado;
- Paradigma hierárquico fracamente distribuído;
- Paradigma hierárquico fortemente distribuído;
- Paradigma cooperativo fortemente distribuído.

O fator chave para esta classificação é o número de gerentes e de agentes presentes na rede, assim como características das relações entre os mesmos. Enquanto sistemas centralizados possuem apenas um gerente, sistemas distribuídos possuem mais de um. A diferença entre sistemas fracamente distribuídos e fortemente distribuídos é que, no caso daqueles, o número de gerentes é bem pequeno quando comparado ao número de agentes, normalmente não ultrapassando uma ou duas dezenas. A distinção entre sistemas hierárquicos e cooperativos, por sua vez, reside no tipo de interação entre gerentes e agentes, em particular no que diz respeito ao tipo de *delegação* que ocorre entre os mesmos. O conceito de delegação será examinado a seguir.

Em sistemas distribuídos, é comum que elementos que desempenhem uma atividade em conjunto solicitem uns aos outros a execução de determinadas tarefas. Isto pode ser feito, entre outras razões, para que as tarefas sejam desempenhadas pelos elementos mais apropriados (no caso de haver elementos especializados em um determinado tipo de tarefa) ou para que haja um balanceamento da carga de processamento entre os diversos elementos do sistema. Este processo de passagem da responsabilidade pela execução de uma tarefa de um elemento a outro recebe a denominação de *delegação* [MartinFlatin99]. Podem-se identificar dois tipos principais de delegação, a saber: a *delegação vertical* e a *delegação horizontal*. Quando a delegação ocorre entre um gerente e um agente, ou entre gerentes de

Tabela 2.6: Relação entre paradigmas e tecnologias de gerência

Paradigma	Tecnologia / Protocolo
Centralizado	SNMPv1
Hierárquico fracamente distribuído	SNMPv2 (com a MIB gerente-gerente) e OSI/TMN
Hierárquico fortemente distribuído	Código móvel e objetos distribuídos
Cooperativo fortemente distribuído	Agentes inteligentes

diferentes níveis hierárquicos, se tem a delegação vertical. Caso a delegação ocorra entre gerentes (ou mesmo agentes) de mesmo nível, ocorre a delegação horizontal. Assim sendo, seguem o paradigma hierárquico sistemas em que a delegação vertical é predominante sobre a horizontal. Caso o inverso seja observado, os sistemas em questão seguem o paradigma cooperativo.

Para a melhor compreensão destes paradigmas, a Tabela 2.6 exemplifica quais tecnologias (ou protocolos) implementam cada um dos paradigmas de gerência aqui expostos. As principais tecnologias representantes dos paradigmas centralizado e fracamente distribuído foram discutidas na Seção 2.1.2. As tecnologias dos paradigmas fortemente distribuídos serão apresentadas a seguir.

A arquitetura tradicional de gerência de redes, amplamente utilizada pela indústria nos dias de hoje, é fortemente baseada em soluções centralizadas (e também soluções hierárquicas fracamente distribuídas) de controle e gerência de serviços e equipamentos. Uma forte evidência deste fato é que os protocolos mais comuns para gerência de redes de telecomunicações ainda são o SNMP e o CMIP (que seguem os paradigmas centralizado e fracamente distribuído). A motivação para a criação destes protocolos foi resolver a questão da interoperabilidade entre diferentes fabricantes, e não houve, pelo menos inicialmente, uma preocupação em se adotar modelos de gerência distribuídos.

As principais críticas feitas ao modelo centralizado normalmente estão relacionadas a três características que são bastante desejáveis a sistemas de gerência, mas que são difíceis de se atingir devido às características de tal modelo. Estas três características são confiabilidade, escalabilidade e flexibilidade [Schönwälder00].

É difícil atingir-se níveis satisfatórios de confiabilidade em um sistema de gerência baseado no modelo centralizado uma vez que as estações de gerência representam pontos de falhas que podem comprometer todo o sistema se deixarem de operar corretamente. Outra situação em que há o comprometimento da gerência ocorre quando, devido a falhas em equipamentos, parte da rede se torna inalcançável a partir das estações de gerência. Em tais circunstâncias, um

subconjunto dos equipamentos da rede estaria fora do controle da gerência apesar de não haver problemas nas estações de gerência propriamente ditas. Obviamente, é extremamente desejável que redes de telecomunicações continuem em operação mesmo em caso de falhas de alguns de seus componentes.

O modelo centralizado enfrenta dificuldades em relação à escalabilidade por duas razões principais. A primeira é carga computacional exigida das estações de gerência que se tornam responsáveis por todos os equipamentos da rede. Outra é a carga de comunicações entre as estações de gerência e equipamentos. Como os recursos computacionais e de comunicação das estações de gerência são limitados, acaba criando-se um limite para o tamanho máximo da rede que estas estações podem suportar. Este problema é ainda mais grave em situações de falhas, quando se espera que a gerência tenha uma atuação mais intensa para contorná-las ou corrigi-las.

Outra questão é que, em um modelo de gerência centralizado, as funcionalidades de gerência são tipicamente pré-definidas e limitadas. Isto se deve ao fato destes modelos adotarem o paradigma cliente/servidor, em que há uma rígida associação de funcionalidades aos servidores durante a fase de projeto dos mesmos. Assim sendo, não há muitas possibilidades de se adicionar ou customizar funcionalidades de gerência, além, obviamente, do que o fabricante determinou. Esta pequena flexibilidade também se deve às limitações de recursos nas estações de gerência, uma vez que estas funcionalidades extras poderiam impactar ainda mais a escalabilidade do sistema de gerência.

A evolução natural do modelo de gerência centralizada foi o modelo de gerência hierárquico (também conhecido como modelo fracamente distribuído). A principal diferença entre os dois modelos é que não existe apenas uma única estação de gerência responsável por toda a rede. Há duas ou mais estações que dividem entre si a responsabilidade de administrar os diversos elementos e serviços da rede. Este modelo também introduz o conceito do gerente de gerentes. Em um modelo hierárquico, a rede é dividida em domínios e a cada domínio é atribuído um gerente. Cabe, então, a um gerente central controlar os gerentes de domínio. Nesta organização, o gerente central possui um nível de abstração mais elevado. Um exemplo bastante claro da evolução de um modelo centralizado para um modelo hierárquico pode ser observada entre a primeira e a segunda versão do SNMP, particularmente no que diz respeito à introdução da MIB Gerente-Gerente [RFC1451] e do novo serviço *Inform* que permitem a comunicação entre gerentes de diferentes níveis hierárquicos.

A grande vantagem do modelo hierárquico em relação ao modelo centralizado é a sua melhor escalabilidade, já que o limite para o tamanho máximo da rede é naturalmente aumentado devido à existência de várias estações de gerência.

Há ainda algum ganho em relação à confiabilidade do sistema, já que com este modelo não existe mais um ponto de falha único no sistema de gerência. Entretanto, a questão da falta de flexibilidade, apontada como uma deficiência do modelo centralizado, não é resolvida com o modelo hierárquico, uma vez que a relação entre agente e gerente não é alterada. Os agentes continuam se comportando apenas como meros coletores de dados.

Uma evolução ao modelo de gerência hierárquico é o modelo de gerência fortemente distribuído. A principal proposta deste modelo é permitir uma maior flexibilidade nas atividades de gerência de redes. Para que isto seja possível, os agentes passam a desempenhar papéis mais relevantes e ativos no sistema de gerência, não sendo apenas meros coletores de dados, mas também processando os mesmos. Isto contribui ainda mais para a descentralização do sistema de gerência, o que traz benefícios em relação à escalabilidade e à confiabilidade do sistema como um todo. Estudos quantitativos que indicam as vantagens do uso de modelos distribuídos podem ser encontrados em [Liotta99, Chen02]. A principal desvantagem de modelos distribuídos é que estes são bem mais complexos de se implementar que os modelos centralizados e fracamente distribuídos.

Há pelo menos três vertentes principais de sistemas de gerência distribuídos. Estas três vertentes são (i) código móvel, (ii) objetos distribuídos e (iii) agentes inteligentes (ou agentes de software) [MartinFlatin99].

Mobilidade de código é uma técnica para criação de sistemas distribuídos em que o código executável de um programa é transmitido através de uma rede e executado em diferentes nós da mesma. Um estudo detalhado do tema pode ser encontrado em [Fuggetta98]. Nele são identificadas as três principais formas de mobilidade de código, a saber, (i) *avaliação remota*, em que um cliente invoca um serviço ao servidor, passando-lhe não apenas o nome e os parâmetros de entrada do serviço, como também código do serviço para execução; (ii) *código sob demanda*, em que um cliente interessado em desempenhar alguma tarefa busca o código correspondente à tarefa em questão em um servidor e o executa; e (iii) *agente móvel*, em que uma unidade de execução (agente) migra autonomamente de nó a nó na rede, desempenhando sua atividade.

O trabalho que primeiramente explorou o uso de técnicas de código móvel aplicado à gerência de redes foi o de Goldszmidt *et al.* [Goldszmidt95], com o conceito de *Gerência por Delegação* (*Management by Delegation* – MbD). Este trabalho propõe uma extensão ao tradicional paradigma gerente/agente. A novidade proposta é que gerentes enviem a agentes programas para desempenhar tarefas de gerência. Estes programas são incorporados aos processos em execução nos agentes e executados juntamente com processos pré-existentes. Os resultados de tais programas são armazenados e posteriormente enviados aos gerentes. A principal

motivação desta técnica é a racionalizar o uso dos recursos da rede, evitando, por exemplo, que tabelas inteiras de uma MIB tenham que ser periodicamente transmitidas pela rede para a identificação de falhas.

As principais organizações de padronização na área de gerência de redes (ISO, ITU e IETF) trabalharam no sentido de integrar o modelo de Gerência por Delegação a seus padrões. Os principais resultados foram a Script MIB [Schönwälder00] desenvolvida para o modelo de gerência da Internet (SNMP) e o CMIP *Command Sequencer* [X.753] desenvolvido para o modelo OSI/TMN.

Uma segunda vertente de paradigmas hierárquicos fortemente distribuídos é a baseada em tecnologias de objetos distribuídos. Sistemas baseados em objetos distribuídos são sistemas que seguem os paradigmas de orientação a objetos, mas possuem a peculiaridade de que neles os objetos podem estar localizados em processos ou máquinas diferentes. Para que isto seja possível, há uma camada intermediária de software que provê serviços de serialização e transmissão de objetos entre processos ou máquinas, e serviços de descoberta de objetos remotos, entre outros. Algumas das principais tecnologias que permitem o uso de objetos distribuídos em sistemas são CORBA [OMG04], RMI [Sun] e DCOM [Microsoft].

Destas tecnologias de objetos distribuídos, a que alcançou a maior popularidade no meio de gerência de redes foi CORBA. É importante ressaltar que CORBA foi amplamente utilizada no contexto da TMN, particularmente na implementação de OSFs nas camadas de negócio e serviço (BML e SML) [Redlich98]. No entanto, este tipo mais comum de uso de CORBA em sistemas de gerência de redes não pode ser considerado como fortemente distribuído, uma vez que a relação entre gerentes e agentes permanece equivalente aos sistemas TMN tradicionais. O uso do CORBA só representa de fato uma evolução no sentido de um modelo fortemente distribuído quando os agentes são capazes de desempenhar papéis mais ativos na gerência da rede, garantindo maior flexibilidade ao sistema. Um exemplo de uso de CORBA em sistemas de gerência fortemente distribuídos pode ser encontrado em [Lazar97].

Houve um importante esforço no sentido de prover a interoperabilidade de sistemas legados baseados em CMIP e SNMP com a então emergente tecnologia CORBA. A força tarefa JIDM (*Joint Inter-Domain Management*) [JIDM] foi criada com objetivo de desenvolver os mecanismos para possibilitar esta interoperabilidade. Um dos principais resultados deste esforço foi o mapeamento dos modelos de informação dos modelos de gerência OSI e da IETF (baseados em ASN.1, GDMO e SMI) em estruturas de dados no formato CORBA IDL. Outro importante resultado foi a definição de interfaces CORBA IDL que implementassem todas as interações possíveis em CMIP e SNMP [JIDM00]. Estes dois resultados garantiram os mapeamentos estático e dinâmico (respectivamente) entre os modelos

OSI/TMN e IETF com a tecnologia CORBA e possibilitaram a interoperabilidade entre sistemas baseados nestes modelos de gerência.

Também é importante destacar a iniciativa da ITU e ISO de modernizar o padrão de gerência OSI/TMN através da adoção do paradigma de objetos distribuídos. O resultado destes esforços foi a ODMA (*Open Distributed Management Architecture*) [X.703]. No contexto da ODMA, não há mais gerentes e agentes com papéis fixos, o que caracteriza os paradigmas centralizado e fracamente distribuído. Pelo contrário, qualquer *objeto computacional* (entidade que substitui gerentes e agentes) pode possuir interfaces para gerenciar outros objetos computacionais – assumindo um papel de gerente – ou expor interfaces para permitir que ele seja gerenciado – assumindo um papel de agente.

Por fim, existem ainda sistemas de gerência baseados em um paradigma cooperativo, tipicamente utilizando agentes inteligentes (ou agentes de software). Neste contexto, o termo "agente" possui a conotação usada pela comunidade de inteligência artificial. O grande diferencial que o paradigma de agentes inteligentes apresenta em relação aos paradigmas de código móvel e objetos distribuídos é o fato que nele cada agente é pró-ativo no sentido de atingir determinada meta que lhe foi estabelecida. Esta característica torna mais natural a delegação horizontal de responsabilidades entre os agentes. As principais características de agentes inteligentes e sua aplicação no domínio de gerência de redes serão abordadas na Seção 2.2 e no Capítulo 5, respectivamente.

2.1.4 Tendências em Gerência de Redes

Durante muito tempo tem havido esforços para o desenvolvimento de técnicas e modelos para gerência de redes. As seções anteriores apresentaram alguns dos resultados mais relevantes e difundidos obtidos até o momento nesta área. Mas há também novas técnicas e modelos que representam as tendências para a gerência das novas gerações de redes. O uso de agentes de software é uma destas tendências. Outras duas tendências são sucintamente apresentadas a seguir.

Gerência Baseada em Políticas

O conceito central da gerência baseada em políticas [Strassner03] é utilizar um conjunto de regras (ou políticas) de alto nível para determinar o comportamento geral da rede. Obviamente deve haver mecanismos que mapeiem estas políticas abstratas nas respectivas ações práticas sobre os recursos de rede. A grande vantagem desta estratégia é permitir que operadores de rede especifiquem operações de gerência em termos dos objetivos que precisam ser alcançados, ao invés de ter

que descrever detalhadamente todas as operações que precisam ser executadas. Dois exemplos de trabalhos nesta área são [Yemini00, Verma02].

Há uma interessante relação entre gerência baseada em políticas e o uso de agentes de software para gerência de redes. Isto se deve ao fato de que políticas podem ser diretamente mapeadas em metas que agentes de software deliberativos devam alcançar. Isto permite um casamento entre gerência baseada em políticas e agentes de software. Esta estratégia é usada, por exemplo, em [Kohli03].

Gerência Baseada em XML

Outra tendência que tem ganhado força na área de gerência de redes é o uso de tecnologias baseadas em XML [XML]. Tal movimento tem sido identificado e discutido em trabalhos como [Schönwälder03, Pras04]. Podem-se citar algumas razões para este movimento. A principal razão é que as atuais tecnologias de gerência de redes não satisfazem todos os requisitos desejados por operadores de rede. Logo, novas alternativas estão sendo buscadas. Por outro lado, há uma significativa disseminação e uma ubíqua aplicação de tecnologias baseadas em XML no domínio de tecnologia de informação (TI) e na Internet. Muitos dos resultados destes esforços acabam sendo aplicáveis em domínios tais como o de gerência de redes.

Um ponto particularmente significativo nesta discussão é que algumas das características inerentes à tecnologia XML são bastante interessantes do ponto de vista de gerência de redes. Em [RFC3535] são apresentadas algumas vantagens do XML em relação às atuais tecnologias de gerência de rede, em particular o SNMP. Alguns dos pontos positivos do XML para gerência de redes salientados neste documento são (i) o fato de XML ser um formato de máquina facilmente processável e (ii) que conta com excelente suporte e disponibilidade de ferramentas e bibliotecas para sua manipulação. Além disso, (iii) a linguagem permite a modelagem de estruturas de dados com diferentes níveis de complexidade, e (iv) conta com excelentes mecanismos para definição de novos modelos de dados (DTD [DTD] e XSD [XSD]).

Um relevante exemplo de tecnologia XML aplicada à gerência de redes são os Web Services [WS]. Dois trabalhos que exploram este tema são [Pavlou04, Boutaba04]. Outros exemplos de aplicação de XML para gerência de redes são os padrões WBEM desenvolvidos pela DMTF mencionados anteriormente.

Por fim, uma promissora iniciativa da IETF foi o desenvolvimento do protocolo Netconf [RFC4741, Choi04], que é baseado em XML. Este protocolo tem sido indicado por alguns como um possível sucessor do protocolo SNMP. O Netconf permite não apenas o monitoramento do estado de equipamentos de rede, mas oferece também um mecanismo direto e flexível para a manipulação de

configurações de equipamentos de rede. A ausência de tais mecanismos é apontada como uma grave deficiência do SNMP. O Netconf foi influenciado por protocolos proprietários tais como o JUNOScript [Juniper].

2.2 Agentes de Software

2.2.1 Introdução

Com o desenvolvimento da ciência da computação, em particular a engenharia de software e as tecnologias da Internet, tem havido uma tendência à adoção de arquiteturas distribuídas e descentralizadas para o desenvolvimento de sistemas de software [Emmerich02, Papazoglou03]. Há desafios crescentes no que diz respeito à interoperabilidade de sistemas, à necessidade de operação em ambientes heterogêneos e ao percebido aumento de complexidade dos sistemas de hardware e software. Neste contexto, há quem diga que os modelos mais tradicionais de desenvolvimento de software, em particular as técnicas de orientação a objetos, têm limitações e que novos paradigmas devem ser buscados para enfrentar estes desafios [Zambonelli03]. Um paradigma relativamente novo que tem obtido destaque nos meios acadêmico e industrial e que tem potencial para resolver alguns destes problemas é a engenharia de software baseada em agentes de software [Luck04].

Nesta seção, alguns importantes aspectos acerca de agentes de software serão discutidos. Serão abordadas as características que distinguem agentes de software de outras abstrações existentes na área de engenharia de software, assim como características e arquiteturas de Sistemas Multi-Agentes (SMA). Por fim, serão abordados alguns padrões existentes para a implementação de SMA, um dos quais é empregado neste trabalho.

2.2.2 Características de Agentes de Software

Um agente de software pode ser entendido como um programa independente, que é capaz de agir de forma *autônoma* com o objetivo de atingir metas pré-estabelecidas. O conceito de autonomia neste contexto diz respeito ao fato de que um agente deve ser capaz de agir sem a intervenção direta de seres humanos ou mesmo de outros agentes. Além disso, ele deve ter controle direto e exclusivo sobre seu estado e comportamento.

Este modelo é particularmente diferente, por exemplo, do modelo de orientação a objetos, em que também há um encapsulamento de estado e de comportamento por parte dos objetos. Pode-se dizer que um objeto tem controle

de seu estado, uma vez que suas propriedades não são visíveis nem acessíveis a entidades externas. Isto também vale para agentes. O comportamento de objetos também é de certo modo controlado por estes uma vez que as possíveis mudanças de estado por que eles podem passar e as eventuais passagens de mensagens a outros objetos que eles podem vir a executar estão diretamente descritas e mapeadas em seus métodos. O que um objeto não controla é o momento em que seus métodos são invocados e executados. Sob este ponto de vista, um objeto não tem controle completo de seu comportamento. Seus métodos são invocados com o objetivo de atingir as metas das entidades que os chamam. Isto não ocorre com agentes, uma vez que suas ações não estão diretamente sujeitas à vontade ou influência de entidades externas. Por conta disto, pode-se dizer que, enquanto um objeto é passivo, um agente de software é ativo.

Outra característica interessante sobre agentes de software, e que é abordada na discussão sobre agentes feita por Wooldridge em [Wooldridge02], é o fato de que um agente deve estar sempre situado em um ambiente, que pode ser tanto real quanto virtual. Esta noção da presença de um agente em um ambiente é importante uma vez que remete para o comportamento *reativo* comum a agentes de software. O ambiente influencia o agente no sentido de que este age em função do que percebe ou observa do ambiente em que se encontra. Além disso, suas ações têm o objetivo de influenciar ou alterar o ambiente para que suas metas sejam alcançadas.

As questões da autonomia e reatividade são aspectos chaves na definição do que é um agente de software. Todavia, há diversos exemplos de sistemas autônomos e reativos que geralmente não são associados a agentes de software. Enquadram-se neste grupo os sistemas de controle que monitoram um ambiente do mundo real e que desempenham alguma ação para modificá-lo quando necessário. São representantes deste grupo os sistemas para automação predial e os sistemas de controle de vôo de aeronaves, por exemplo. Outro tipo de sistema autônomo e reativo é aquele em que se encontram os processos *daemon* de um computador. Tais processos monitoram um ambiente de software e também desempenham ações quando este sofre alguma alteração crítica. Um exemplo de processo *daemon* particularmente comum em gerência de redes são os agentes SNMP discutidos na Seção 2.1.

Esta constatação indica que há outros aspectos além da autonomia e reatividade que definem a essência de um agente. Destes aspectos, o mais significativo é provavelmente a pró-atividade. Este aspecto está intimamente ligado à capacidade de um agente de tomar a iniciativa de alguma ação que o ajude a atingir suas metas. Idealmente, um agente de software não tem todo o seu comportamento pré-determinado de forma rígida pelo programador que o criou. Em agentes deliberativos, é comum haver um conjunto de planos pré-determinados

de onde os agentes podem selecionar e executar aqueles que, de acordo com sua percepção e avaliação, o levem em direção aos seus objetivos.

Outro aspecto que diferencia agentes de software dos sistemas autônomos tradicionais é a sociabilidade. Por sociabilidade, entende-se a capacidade de um agente interagir com outras entidades externas, sejam elas outros agentes ou seres humanos. Ao interagir com outras entidades um agente tem como objetivo tanto a busca por recursos que possam ajudá-lo em seus próprios interesses assim como a cooperação com demais entidades que buscam resolver seus próprios problemas.

A lista a seguir, adaptada de [Wooldridge95], sumariza os mais importantes atributos de agentes de software aqui discutidos, e que os distinguem de outros paradigmas existentes na engenharia de software.

Autonomia: agentes de software operam com um mínimo de intervenção externa, seja de outros agentes ou de seres humanos, e, além disso, mantêm controle sobre o seu próprio estado e comportamento;

Sociabilidade: agentes de software podem se comunicar com outros agentes ou com seres humanos com a finalidade de trocarem informações ou cooperarem para atingirem objetivos comuns;

Reatividade: agentes de software percebem o ambiente em que operam e podem tomar ações em função de mudanças que venham a acontecer;

Pró-atividade: agentes de software devem agir orientados a metas, tomando a iniciativa através de planos e ações que os conduzam a seus objetivos.

Estes quatro atributos são freqüentemente considerados o mínimo necessário para que uma entidade de software seja considerada um agente. No referido trabalho estes atributos compõem o que os autores chamam de uma "fraca noção de agência". Não há, entretanto, um consenso universal sobre esta questão, visto que alguns autores excluem a pró-atividade da lista dos atributos essenciais de um agente. Desta forma, é possível definir entidades de software não deliberativas como agentes de software. Por outro lado, há atributos adicionais comumente encontrados e associados a agentes de software. Tais atributos podem ser desejáveis ou mesmo necessários em determinadas aplicações baseadas em agentes de software. A lista a seguir sintetiza alguns dos principais atributos adicionais de agentes de software [Franklin96].

Adaptabilidade: agentes de software podem traçar diferentes planos de ação para atingirem suas metas baseados nas condições do ambiente em que operam, e trocá-los em caso de mudanças no ambiente;

Aprendizagem: agentes de software podem possuir mecanismos de aprendizagem que os permitam adaptar-se a mudanças no ambiente em que atuam;

Mobilidade: agentes de software podem mudar dinamicamente o hospedeiro (*host*) em que executam, mantendo o seu estado, caso esta mudança seja benéfica para seus interesses.

2.2.3 Sistemas Multi-Agentes

Domínios tais como telecomunicações, gerência de processos de negócios, controle industrial, entre outros, são caracterizados pela complexidade dos sistemas neles existentes. A engenharia de software tem estudado e proposto métodos e procedimentos que permitam um melhor gerenciamento desta complexidade. Técnicas de análise e projeto de sistemas comumente aplicam a decomposição de problemas complexos em subproblemas menores, assim como a abstração do sistema através do uso de modelos simplificados e a organização dos diversos componentes do sistema de acordo com suas funcionalidades e dependências.

Muitos autores têm sugerido o uso de agentes de software como uma forma de lidar com tais questões. Neste contexto, os sistemas seriam compostos por diversos agentes, cada um com sua funcionalidade ou papel específico, formando os chamados sistemas multi-agentes (SMA). Cada agente de tais sistemas teria um ou mais objetivos próprios a alcançar e dependeria de outros agentes para tal. Idealmente, tais sistemas multi-agentes seriam capazes de convergir para os resultados esperados pelos usuários.

Segundo Jennings [Jennings01], a decomposição em agentes de software é um modo efetivo de particionar os problemas relativos a um sistema complexo. Ele também sustenta que as abstrações da chamada orientação a agentes são um meio natural de se modelar sistemas complexos. Por fim, ele defende que a filosofia de relacionamentos entre agentes é apropriada para se lidar com as dependências e interações que existem em sistemas complexos.

Alguns dos benefícios citados na literatura sobre o uso de sistemas multi-agentes são citados a seguir [Hayzelden99b]:

- Resolução de problemas que sejam grandes demais para uma única entidade (ou agente) centralizada;
- Redução de custos de processamento (é mais barato em termos de hardware usar vários processadores de desempenho mediano do que usar um único processador com um desempenho equivalente);
- Viabilização da interconexão e interoperabilidade entre sistemas legados existentes;

– Melhorias de escalabilidade, uma vez que a estrutura organizacional dos agentes pode ser alterada dinamicamente de forma a refletir mudanças em seu ambiente;
– Provisão de soluções distribuídas para problemas inerentemente distribuídos;
– Provisão de soluções para problemas cujas fontes de dados são distribuídas.

Além disso, sistemas que implementam o paradigma SMA são mais facilmente escaláveis devido a pelo menos três fatores. Um deles é a (i) modularidade que o paradigma naturalmente impõe, visto que cada agente pode ser entendido como um módulo do sistema que desempenha determinada funcionalidade. Outro fator é o (ii) fraco acoplamento entre seus componentes, uma vez que os agentes são entidades relativamente independentes entre si. Um último fator é o uso de (iii) abstrações de mais alto nível (o nível de abstração de um agente é maior que o de um objeto, por exemplo). Tais vantagens são extremamente convenientes para a construção de sistemas de gerência de redes como é o caso da arquitetura proposta neste trabalho. Estas vantagens são particularmente pronunciadas no que diz respeito à escalabilidade e à flexibilidade de tais sistemas.

2.2.4 Arquiteturas de Sistemas Multi-Agentes

Sistemas multi-agentes geralmente são considerados sistemas com um alto nível de complexidade devido ao fato de serem compostos por diversos componentes (agentes) que podem se relacionar – para efeitos práticos – de forma imprevisível ou indeterminada. Para lidar com sistemas complexos é comum o uso de modelos abstratos que permitam uma visão mais geral do sistema em questão. Tais modelos são muitas vezes denominados arquiteturas.

No caso de SMA, é possível identificar dois níveis de arquitetura. O primeiro diz respeito à modelagem dos componentes que formam a estrutura interna de um agente. Esta arquitetura pode ser denominada *intra-agente*. Um segundo nível de arquitetura diz respeito aos relacionamentos e interações entre agentes em um SMA. Esta arquitetura pode ser denominada *inter-agente* [Hayzelden99b].

Em [Wooldridge95] são identificados três tipos diferentes de arquiteturas intra-agentes, a saber, a arquitetura de agentes reativos, a arquitetura de agentes deliberativos e a arquitetura de agentes híbridos. Agentes reativos são aqueles que mantêm nenhuma ou pouca informação sobre o estado do ambiente em que se localizam. Estes agentes geralmente agem apenas em função de estímulos externos e seu comportamento é baseado em regras que vinculam uma determinada condição observada no ambiente a uma ação correspondente.

Agentes deliberativos, por sua vez, mantêm uma rica representação do estado do ambiente em que se encontram. Este conhecimento sobre o ambiente é utilizado

para o planejamento de ações futuras. Por estas razões, diz-se que estes agentes mantêm um *estado mental*. Estes agentes geralmente analisam diversas linhas de ação (ou planos) com o objetivo de identificar as ações que podem levar o ambiente para o estado que desejam. Estas características tornam estes agentes mais complexos que os agentes reativos, porém também os conferem maior flexibilidade. Um dos modelos mais utilizados para a implementação de agentes deliberativos é o modelo BDI (*Belief, Desire and Intention*) [Bratman87]. Este modelo é baseado nos conceitos de crenças, desejos e intenções. O conjunto de crenças representa o estado que o agente mantém sobre o ambiente. Desejos são os objetivos que o agente pretende eventualmente alcançar. Intenções são os desejos que o agente está comprometido em satisfazer através de planos em execução.

Por fim, agentes híbridos são aqueles que conjugam as características tanto de agentes reativos quanto de agentes deliberativos. Eles conjugam o mecanismo de resposta rápida baseado em estímulos externos assim como o comportamento pró-ativo possibilitado pela sua capacidade de planejamento.

No que diz respeito a arquiteturas inter-agentes, há pelo menos duas grandes vertentes, a de agentes cooperativos e a de agentes individualistas. Agentes cooperativos agem em conjunto para atingir um objetivo comum. Agentes individualistas agem em benefício próprio buscando satisfazer prioritariamente seus próprios objetivos. Um agente individualista somente coopera com outro agente caso receba algo de valor em troca.

Muitos esforços têm sido despendidos no estudo de agentes colaborativos. Algumas das principais razões citadas em [Hayzelden99b] para a distribuição de tarefas entre agentes colaborativos são: (i) a redução de custos de comunicação associados a uma entidade central, (ii) a melhoria de desempenho através de paralelismo de atividades, (iii) o ganho em reatividade devido a não necessidade de consulta a uma entidade central, e (iv) o ganho em confiabilidade devido a não dependência de um elemento central único.

A aplicação de agentes individualistas, por sua vez, está geralmente restrita a casos em que agentes que representam diferentes grupos ou organizações interagem entre si em ambientes abertos. É comum que esta interação se dê através de negociações em que um agente busca um serviço com o agente que lhe forneça as melhores condições. A implementação de SMA com agentes individualistas é mais difícil, pois há importantes aspectos de segurança que devem ser levados em consideração. Além disso, cada agente (ou o ambiente em que os agentes se encontram) deve estar preparado para lidar e se proteger de agentes "mal-intencionados".

2.2.5
Padrões de Sistemas Multi-Agentes

A tecnologia de sistemas multi-agentes (SMA) ainda é pouco utilizada fora do meio acadêmico e pode ser considerada relativamente imatura quando comparada com tecnologias mais tradicionais de engenharia de software. Isto é natural na medida em que ela é fruto de pesquisas recentes e ainda não teve alguns importantes problemas equacionados. Espera-se, entretanto, que esta tecnologia amadureça e se torne mais popular nos próximos anos [Nwana99a, Luck04].

Um fator que até pouco tempo dificultava uma adoção mais ampla da tecnologia de SMA era o fato de não haver padrões universalmente reconhecidos contendo especificações com os requisitos mínimos para a implementação de SMA e que garantissem a interoperabilidade entre diferentes agentes de software.

Para remediar esta situação importantes iniciativas no sentido de padronização têm sido tomadas. Uma das primeiras e mais influentes iniciativas foi a *Knowledge Sharing Effort* (KSE) iniciada por volta de 1990 pela DARPA [Neches91]. Seu objetivo era desenvolver técnicas, metodologias e ferramentas que permitissem o compartilhamento e a reutilização de conhecimento em sistemas de software. Apesar de esta iniciativa não ser focada exclusivamente em agentes de software, muitos de seus resultados foram aproveitados por esta comunidade. Um dos resultados mais importantes deste esforço, e que teve uma ampla utilização em SMA, foi a linguagem *Knowledge Query and Manipulation Language* (KQML) [Finin97].

A linguagem KQML permite a comunicação entre dois agentes de software de forma padronizada. Ela apresenta um alto nível de abstração, é orientada à troca de mensagens, e permite a troca de informações de forma independente da sintaxe ou ontologia usadas para representar o conteúdo das mensagens. A linguagem KQML pode ser dividida em três camadas, a saber, a camada de conteúdo, a camada de comunicação, e a camada de mensagem. A camada de conteúdo é a que traz o conteúdo da mensagem propriamente dito. A camada de comunicação contém parâmetros de nível básico da comunicação como, por exemplo, as identidades do remetente e destinatário e um identificador único relativo à comunicação. Por fim, a camada de mensagem determina o tipo de interação entre os agentes. Sua principal função é identificar o protocolo de rede a ser utilizado e o tipo da mensagem transmitida. A KQML é baseada em performativas da teoria de atos da fala [Searle69]. Isto permite que o remetente transmita sua intenção (informar, solicitar informação ou solicitar uma ação, por exemplo) quando do envio da mensagem [Labrou99].

A linguagem utilizada para comunicação entre dois agentes de software talvez seja o ponto mais crítico para permitir a interoperabilidade entre os mesmos.

Agentes que se comunicam precisam necessariamente de uma linguagem comum e mutuamente inteligível para que a troca de informações seja viabilizada. Além da linguagem de comunicação, também é necessário especificar e padronizar outros aspectos da comunicação entre agentes. Aspectos bastante pragmáticos como o registro da existência (e da saída) de um agente em um ambiente, e mecanismos para que se encontre um agente em particular neste ambiente (serviços de diretório e nome) devem ser padronizados para permitir a interoperabilidade entre sistemas. A padronização destas e de outras questões relativas à implementação de SMA tem sido o objetivo da organização FIPA (*Foundation for Intelligent Physical Agents*) desde a sua fundação em 1996.

Os padrões da FIPA não têm como objetivo determinar ou especificar a arquitetura interna dos agentes, mas sim prover interfaces que permitam a comunicação entre agentes. Os padrões da FIPA podem ser divididos em cinco grupos principais de acordo com seu assunto, a saber, arquitetura abstrata, transporte de mensagens, gerência de agentes, comunicação de agentes e aplicações de agentes [Dale01].

O propósito da arquitetura abstrata da FIPA [FIPA00001] é garantir interoperabilidade e reusabilidade. Para tanto são identificados os principais elementos necessários para a criação de SMA, assim como as relações entre estes elementos. A partir desta descrição abstrata de elementos e relacionamentos é possível construir implementações concretas e distintas de SMA que sejam compatíveis entre si uma vez que compartilham os mesmos conceitos e abstrações básicos.

O transporte de mensagens lida com o envio e a representação de mensagens através de diferentes protocolos de redes no contexto de uma plataforma de agentes. Para a FIPA, uma mensagem é composta por um envelope e um corpo. O envelope contém informações básicas (como remetente e destinatário) necessárias para que a plataforma entregue a mensagem de forma correta. O corpo da mensagem contém o conteúdo da mesma geralmente codificado na linguagem FIPA ACL [FIPA00061]. A FIPA padroniza o suporte para o serviço de envio de mensagens em uma plataforma de agentes [FIPA00067], diretrizes para o uso de protocolos tais como IIOP [FIPA00075], HTTP [FIPA00084] e WAP [FIPA00076], representações de envelopes de mensagens em XML [FIPA00085] e em formato binário [FIPA00088], assim como representações da linguagem FIPA ACL em formato texto [FIPA00070], XML [FIPA00071] e binário [FIPA00069]. Com estes padrões é possível obter-se uma interoperabilidade para a troca de mensagens entre agentes.

Outro importante padrão da FIPA é o que diz respeito à gerência de agentes. A especificação [FIPA00023] provê a arquitetura do ambiente em que

os agentes existem e operam. Nela são estabelecidos os modelos para criação, registro, localização, comunicação, migração e remoção de agentes. Dois dos mais importantes componentes aqui definidos são o serviço de diretório (*Directory Facilitator*) que permite o registro e a localização de serviços oferecidos por agentes e o sistema de gerência de agentes (*Agent Management System*) que controla os agentes em uma determinada plataforma assim como alguns aspectos do ciclo de vida dos mesmos.

Outro conjunto de padrões FIPA diz respeito à comunicação entre agentes. Ao contrário dos padrões relacionados ao serviço de transporte de mensagens, que lidavam com aspectos de baixo nível relacionados ao envio e representação das mensagens, os padrões de comunicação entre agentes definem um rico modelo semântico baseado na teoria de atos da fala [Searle69] para permitir comunicações de mais alto nível de abstração entre agentes. Através das chamadas performativas (tais como requisitar, confirmar ou informar) definidas em [FIPA00037], é possível que o agente que receba uma mensagem identifique melhor o contexto em que a mensagem deve ser interpretada. Outro padrão relacionado à comunicação entre agentes é a linguagem FIPA ACL [FIPA00061]. Esta linguagem provê mecanismos para adicionar contexto a uma mensagem, tais como a performativa (como indicado acima), remetente e destinatário e o protocolo de interação da mensagem. A linguagem FIPA ACL foi fortemente influenciada pela linguagem KQML apresentada anteriormente. Por fim, há ainda um conjunto de padrões que definem os chamados protocolos de interação, que nada mais são do que descrições de tipos de conversações comuns entre agentes. Dois exemplos de protocolos de interação são a interação de requisição [FIPA00026] em que um agente requisita que outro agente desempenhe alguma ação e a interação de consulta [FIPA00027] em que um agente solicita a outro agente a permissão para desempenhar alguma ação.

Por fim, o quinto e último conjunto de padrões especifica quatro aplicações baseadas em agentes. Também são descritos os serviços e as ontologias usadas em cada aplicação. Um dos casos especificados diz respeito ao provisionamento automático de serviços de rede para aplicações multi-mídia [FIPA00082]. Este exemplo é analisado com mais detalhes no Capítulo 5.

3
A Arquitetura NeMaSA

3.1
Introdução

A arquitetura NeMaSa tem como objetivo principal possibilitar a gerência de falhas em redes legadas de telecomunicações de forma escalável. Geralmente tais redes são controladas e administradas por sistemas centralizados de gerência, muitas vezes seguindo padrões tais como o TMN. Tais sistemas concentram informações essenciais para procedimentos de diagnóstico e correção de falhas. Caso tais procedimentos sejam automatizados e executados com muita freqüência, corre-se o risco de que o funcionamento tanto do sistema de gerência como o do sistema de diagnóstico sejam prejudicados devido à inevitável sobrecarga daquele.

Este trabalho traz como contribuição a proposta de um mecanismo para se contornar este problema de sobrecarga através da distribuição pela rede das informações necessárias para os procedimentos de diagnóstico. Tal distribuição é realizada com o auxílio de agentes de software que extraem as informações do sistema centralizado de forma mais organizada. Desta forma, é possível que os agentes responsáveis por executar os procedimentos de diagnóstico e correção de falhas desempenhem suas atividades sem a necessidade de uma comunicação direta com o sistema centralizado, o que alivia a carga sobre o mesmo.

3.2
Motivação

Uma das principais motivações para a realização do presente trabalho foi proveniente da realidade observada em uma rede de dados de uma das maiores empresas prestadoras de serviços de telecomunicações no Brasil. O núcleo da rede estudada é baseado no protocolo ATM [I.321]. Há, entretanto, outras tecnologias com uma significativa presença na rede, como o Frame Relay [I.233] e o TDM (circuitos de dados baseados em multiplexação determinística). Estas tecnologias enquadram-se no nível dois do modelo OSI e são todas baseadas na técnica de circuito virtual, em que as rotas entre dois nós são definidas preliminarmente e mantêm-se inalteradas até o fim da transmissão de dados. Este modelo diverge significativamente do modelo de datagramas (adotado pelo protocolo IP, por

exemplo), em que as rotas são definidas dinamicamente em cada nó na medida em que os pacotes percorrem a rede.

No caso particular da rede estudada, é muito comum o uso de circuitos virtuais permanentes, em que recursos da rede são reservados de forma permanente para clientes, criando conexões dedicadas entre duas de suas instalações, como, por exemplo, duas filiais de uma empresa em cidades diferentes. A criação e a manutenção destes circuitos são efetuadas através de um sistema centralizado de gerência que abrange a rede em sua quase totalidade. Também cabe a este sistema legado a tarefa de monitorar a rede, detectando falhas em equipamentos e determinando novas rotas para os circuitos afetados. Devido às dimensões desta rede, que abrange boa parte do território brasileiro, este sistema encontra-se sobrecarregado. De fato, alguns tipos de monitoramento menos críticos tiveram de ser desativados para permitir que o sistema pudesse continuar exercendo tarefas mais importantes de gerência de elementos e circuitos.

Um dos principais objetivos da equipe que administra esta rede é o de reduzir o tempo para o diagnóstico de falhas em equipamentos e a descoberta de erros de configuração. Entende-se que esta redução do tempo de diagnóstico resulte em uma redução dos custos operacionais necessários para manter a rede em funcionamento. Para suprir esta necessidade foi desenvolvido um sistema de diagnóstico baseado em heurísticas. Este sistema busca na rede (tanto em equipamentos quanto em sistemas de gerência) os sintomas de problemas mais comuns ou mais críticos, identifica falhas e, quando possível, as corrige.

Um detalhe importante, entretanto, é que, apesar de o sistema desenvolvido ter automatizado o processo de diagnóstico de falhas, ele ainda depende da intervenção de um operador de rede (humano) que solicite explicitamente a execução de testes em um determinado circuito ou equipamento. Por esta razão, um próximo passo no sentido de possibilitar uma redução ainda maior nos custos de operação desta rede seria prover um gerenciamento mais autônomo, com uma menor intervenção direta dos operadores de rede. Desta forma seria possível identificar algumas falhas antes mesmo que elas cheguem a afetar significativamente o usuário.

Para que esse objetivo seja alcançado, é importante monitorar equipamentos e circuitos de forma contínua. Em muitos casos, pode também ser necessário executar testes custosos em termos de processamento. Um possível exemplo de teste custoso seria um em que se acesse uma base temporal para identificar desvios em relação a um comportamento historicamente observado. Tais requisitos, associados à grande dimensão da rede, sugerem que se dê especial atenção à questão da escalabilidade quando da formulação de uma solução para este problema. É neste contexto de busca de uma maior automação nos mecanismos de detecção de

falhas, levando-se em consideração questões de escalabilidade, que se enquadra a arquitetura apresentada neste trabalho.

3.3 Descrição Geral

3.3.1 Principais Desafios

O principal objetivo que se pretende alcançar através da arquitetura proposta neste trabalho é a aplicação de testes em equipamentos para o diagnóstico e correção de falhas na rede. Tais testes devem ser realizados de forma que não haja sobrecarga sobre a rede como um todo, nem sobre nenhum de seus elementos em particular. Um ponto importante é que a arquitetura proposta deve se adequar aos requisitos e às limitações da rede de telecomunicações previamente apresentada.

Muitas das fontes de informação necessárias para o diagnóstico de falhas estão dispersas pela rede. O exemplo mais nítido disto são as informações sobre o estado dos equipamentos que são obtidas diretamente dos próprios equipamentos. Como estas fontes de dados estão distribuídas, é bastante razoável o emprego de alguma técnica também distribuída para a coleta e manipulação destas informações. Uma vantagem desta estratégia distribuída é que ela contribui naturalmente para uma melhor escalabilidade da arquitetura.

Há, entretanto, uma fonte de dados fundamental para o funcionamento da solução aqui proposta que se encontra concentrada em um repositório centralizado. Esta fonte de dados é o sistema de gerência legado previamente citado. As informações mais relevantes que ele contém são a topologia e o inventário da rede. De fato, a maioria das informações que o sistema de diagnóstico de falhas citado anteriormente extrai deste repositório para efetuar seus testes são dados de topologia e inventário da rede. A estratégia de acesso a estes dados representa um primeiro desafio a ser superado, já que informações de topologia e o inventário da rede são essenciais para o diagnóstico de algumas falhas. Por outro lado, o acesso a este sistema de gerência representa um gargalo que pode comprometer a escalabilidade da solução proposta. Em um caso mais extremo, pode-se até mesmo prejudicar o funcionamento de tal sistema de gerência (e, por conseqüência, de toda a rede) caso haja uma demanda muito grande de informações de topologia solicitada simultaneamente. Como será visto a seguir, a solução aqui proposta utiliza um mecanismo para acessar este repositório centralizado de uma forma racional, evitando demandas excessivas sobre o mesmo.

Outro importante desafio deste trabalho é permitir a criação de soluções mais autônomas de gerência de falhas baseadas na arquitetura proposta. Em uma rede de

telecomunicações em produção, o tempo economizado no processo de diagnóstico e correção de uma falha é proporcional a uma redução nos custos de operação da rede. Por esta razão é que a autonomia é tão importante neste domínio. Idealmente, as falhas devem ser identificadas e tratadas antes mesmo de se manifestarem, ou pelo menos, assim que se manifestarem.

Há pelo menos duas formas de se imprimir autonomia na gerência de falhas em redes. Uma forma, simples e universalmente utilizada, é coletar estatísticas e alarmes enviados por equipamentos e tomar ações baseadas nas informações recebidas. Outra forma, mais complexa e até por isso menos comum, é a previsão de falhas baseada em dados coletados da rede. Normalmente um perfil histórico do comportamento da rede é traçado, e este é continuamente comparado com o estado presente da rede. Quaisquer desvios ou diferenças significativas entre o perfil histórico e o estado atual podem indicar a ocorrência de falhas. Em alguns casos, técnicas baseadas em inteligência artificial podem ser aplicadas para efetuar estas comparações.

A grande vantagem do método de monitoramento de estatísticas e alarmes é que as evidências ou sintomas das falhas são descobertos pelos próprios equipamentos, cabendo ao sistema de gerência de falhas apenas a tarefa de filtrá-las, interpretá-las e tomar as medidas corretivas cabíveis. A principal desvantagem deste método é que em muitos casos, os problemas decorrentes das falhas já se manifestaram e afetaram o funcionamento da rede.

A principal vantagem do método de previsão de falhas é que ele torna possível identificar indícios de eventuais falhas antes mesmo que elas venham a ocorrer. Desta forma, medidas corretivas podem ser tomadas antes que o funcionamento da rede venha a ser afetado significativamente. Por outro lado, sua grande desvantagem é a relativa complexidade de implementação dos mecanismos de predição. Tais mecanismos normalmente exigem a criação e manutenção de bases com dados históricos sobre o funcionamento da rede, e aplicam técnicas estatísticas ou mesmo de inteligência artificial.

Um terceiro desafio na concepção da arquitetura proposta é mantê-la suficientemente flexível para que possa ser aproveitada sem maiores alterações em outras redes de telecomunicações semelhantes. Esta não é uma questão fundamental, uma vez que o objetivo principal deste trabalho é atender os problemas da rede previamente apresentada. Entretanto, sempre que possível, é interessante adotar mecanismos que permitam uma fácil adaptação da arquitetura proposta a uma nova rede, com novos tipos de falhas. Como será visto adiante, o mecanismo de definição e execução dos testes permite que diferentes tipos de testes sejam desempenhados na rede através da arquitetura proposta.

3.3.2
Soluções Propostas

Levando-se em consideração a razoabilidade do uso de técnicas distribuídas e os requisitos de autonomia e flexibilidade, o uso de agentes de software neste problema é bastante apropriado, conforme pode ser depreendido da discussão das seções anteriores sobre o uso de agentes de software no domínio de gerência de redes.

Por essa razão, foi concebido um modelo baseado em agentes de software distribuídos pela rede, que executam testes em equipamentos através de protocolos comuns de gerência de redes, agindo diretamente sobre os nós ou sistemas de gerência. Esta solução permite uma ação mais intensa sobre os equipamentos do que seria possível com um modelo centralizado. Além disto, esta solução também permite distribuir a carga de processamento de testes mais custosos em diversas máquinas, uma vez que os agentes de software podem agir de forma independente entre si.

Outra vantagem no uso de agentes de software é que eles representam um meio bastante apropriado para a criação de sistemas autônomos e pró-ativos. Apesar de o aspecto de pró-atividade na arquitetura proposta não ser profundamente abordado neste trabalho, há perspectivas de futuros desenvolvimentos nesta área (vide o Capítulo 6).

A principal desvantagem do uso de agentes de software é que a arquitetura proposta se torna moderadamente complexa. Há diversas entidades (agentes) agindo de forma relativamente independente, há uma maior complexidade nas interações entre estas entidades, o que resulta em mais possíveis formas de o sistema falhar e não atingir seus objetivos. Contudo, considera-se que as vantagens deste modelo baseado em agentes de software sejam fortes o bastante para justificar tal proposta.

A solução proposta para se contornar a questão do acesso ao repositório centralizado foi distribuir pela rede a informação de topologia contida no repositório centralizado. Para isto são utilizados agentes de software projetados especificamente com o propósito de obter e difundir estes dados. Para viabilizar o emprego desta estratégia, a rede pode ser dividida em partições ou sub-redes, que por sua vez também podem ser subdivididas em outras sub-redes, tantas vezes quantas necessárias. A manutenção da informação de topologia de cada uma dessas sub-redes fica a cargo de um agente de software que centraliza nele as solicitações de informações de topologia da sub-rede pela qual é responsável. A Figura 3.1 representa tal organização, ilustrando a decomposição da rede nacional em três sub-redes, e a decomposição da sub-rede do Rio de Janeiro em outras três sub-redes. Esta estratégia de divisão da rede em níveis hierárquicos permite aliviar a carga que inevitavelmente surgiria sobre o sistema de gerência centralizado.

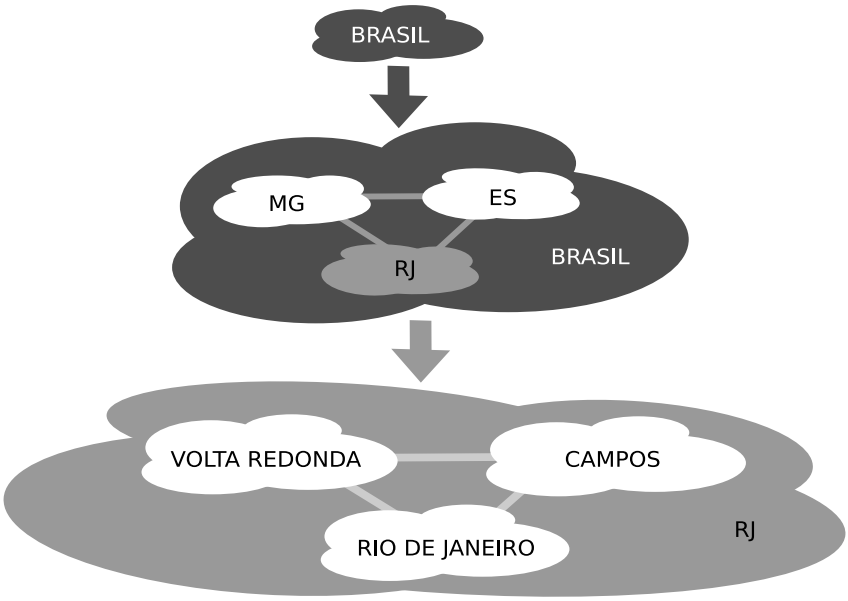

Figura 3.1: Hierarquia de redes

Com esta organização em mente, este trabalho propõe a modelagem do sistema com o uso de quatro tipos de agentes de software, listados a seguir:

Agente de gerência. Serve de ligação entre o sistema de gerência centralizado e os demais agentes. Tem a responsabilidade de controlar os acessos aos recursos do sistema de gerência centralizado, evitando que este seja eventualmente sobrecarregado por requisições dos demais agentes;

Agente de topologia. Tem como principal responsabilidade fornecer informações de topologia de sua sub-rede aos demais agentes (exceto o de gerência). Ele deve implementar mecanismos para manter-se atualizado em relação à topologia real mantida na base do sistema de gerência. Para tanto, ele pode comunicar-se diretamente com o agente de gerência ou através de outros agentes de topologia, valendo-se de consultas diretas às bases de topologia mantidas por estes agentes ou registrando-se para o recebimento de notificações de mudanças de topologia;

Agente de teste. Monitora e age diretamente sobre equipamentos da rede, verificando o estado destes e coletando estatísticas de seu funcionamento. Este comportamento é realizado através de *testes* solicitados por outros agentes;

Agente de usuário. Serve como interface entre o sistema e seus usuários, interpretando os parâmetros de entrada dos usuários, disparando as ações que forem necessárias e retornando aos usuários o resultado destas ações.

A Figura 3.2 ilustra a distribuição desses agentes em uma rede fictícia.

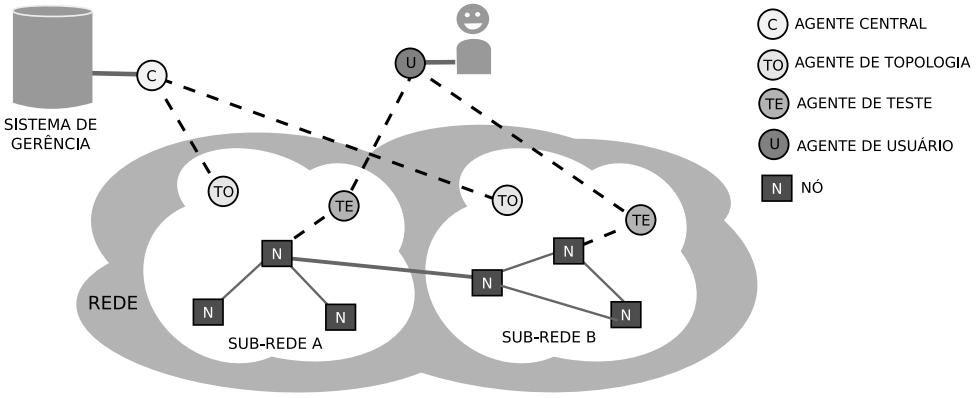

Figura 3.2: Distribuição dos agentes pela rede

Um outro aspecto central da solução proposta diz respeito aos testes que serão feitos nos equipamentos e circuitos da rede. No contexto deste trabalho, um *teste* pode ser entendido como um fluxo de tarefas automatizadas necessárias para a obtenção de um determinado diagnóstico ou correção de falha. Estes testes usualmente coletam dados de equipamentos, processam os mesmos, e fornecem um laudo que identifica uma eventual falha. Naturalmente, os testes que necessitarem de informações de topologia ou inventário da rede para identificar alguma falha, devem consultar o agente de topologia correspondente e extrair dele os dados desejados. Também é possível que um teste aja sobre um equipamento de rede a fim de efetuar uma correção e não apenas uma coleta de dados.

Em alguns casos, pode ser interessante ou até mesmo necessário que um teste consulte um sistema de gerência para buscar alguma informação não relacionada à topologia ou inventário da rede. Apesar de esta comunicação direta entre um agente de teste e o sistema de gerência ser possível, tal estratégia de comunicação direta é desaconselhada, pois pode fragilizar a solução em termos de sua escalabilidade. Contornar este problema no caso de informações de topologia foi, na verdade, uma das maiores motivações para o presente trabalho. Uma melhor solução seria, se possível, adaptar os agentes de topologia para trabalhar com este novo tipo de dado.

Um exemplo simples de teste seria verificar se uma determinada porta está com uma taxa de perda de pacotes acima de um limite aceitável. Neste caso, o teste deve conter tarefas para buscar no equipamento o número de pacotes perdidos na porta e para calcular a taxa em questão. Os parâmetros de entrada deste teste seriam a porta a ser verificada e o limite de tolerância de pacotes perdidos. O resultado do teste seria uma simples mensagem indicando se há um excesso de pacotes perdidos.

Um exemplo mais complexo de teste, que faz uso de informações de topologia, seria verificar um por um os recursos de rede que um determinado

circuito de um usuário da rede utiliza. Este cenário é comum em redes ATM, Frame Relay e TDM em que os circuitos utilizados por cada usuário podem ser determinados em um momento inicial e podem permanacer inalterados indefinidamente. Nestes casos, um teste interessante é percorrer cada recurso que compõe este circuito verificando a existência de anormalidades que porventura possam prejudicar a qualidade do serviço prestado ao usuário em questão. Outro exemplo interessante de teste em que informações de topologia de rede se fazem necessárias é o teste de fonte de sincronismo de nós. Este teste é discutido em detalhes no Capítulo 4.

As tarefas que compõem um teste e que se comunicam com equipamentos podem ser classificadas como *sensores* ou *comandos*. Um sensor é uma tarefa que busca de um equipamento alguma informação relativa ao estado do mesmo. Um comando é uma tarefa que executa alguma ação sobre um equipamento, possivelmente alterando suas configurações. Tanto sensores quanto comandos atuam como mecanismos de comunicação entre os agentes de teste e os equipamentos. A diferença entre ambos é que um comando é, por definição, intrusivo e pode mudar o comportamento de um equipamento. Um sensor, por sua vez, apenas lê dados do equipamento sem maiores efeitos colaterais. Tal distinção é importante porque permite identificar os testes intrusivos (que são potencialmente mais perigosos que os testes que apenas possuem sensores). Outro aspecto relacionado é que um teste que realiza uma correção possui necessariamente pelo menos um comando.

Sensores e comandos devem realizar a comunicação com os equipamentos usando protocolos de gerência de rede tais como SNMP [RFC3411], TL1 [GR-831], Netconf [RFC4741] e JUNOScript [Juniper]. Uma outra forma de acesso comum é a CLI (*Command Line Interface*), normalmente conduzida através dos protocolos Telnet [RFC854] ou, preferencialmente, SSH [RFC4251].

Para garantir uma maior flexibilidade à arquitetura proposta, buscou-se manter um baixo acoplamento entre o agente de teste e os tipos de teste que podem ser aplicados à rede. A implicação prática disto é que o agente de teste carrega dinamicamente os testes a partir de uma base de testes. Uma vantagem desta estratégia é que ela facilita a criação de novos tipos de testes, uma vez que o restante do sistema (em particular o agente de teste) não precisa ser alterado. Outra conseqüência é que também é facilitada a adaptação desta arquitetura para funcionamento em uma outra rede de telecomunicações, uma vez que um novo conjunto de testes direcionados a outra rede pode ser adaptado à arquitetura aqui proposta.

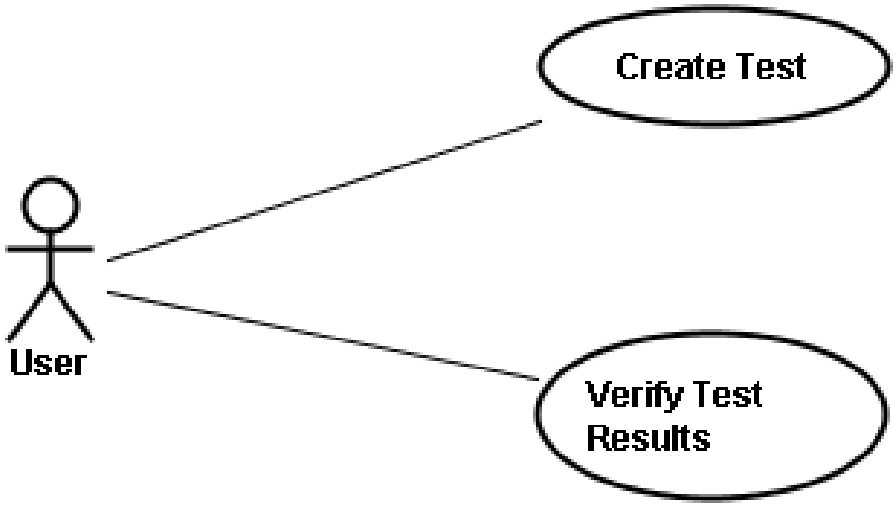

Figura 3.3: Diagrama de casos de uso

3.4 Arquitetura

3.4.1 Requisitos Funcionais

Sob a ótica do usuário final, geralmente um operador de rede, a funcionalidade do sistema proposto é bastante simples. O que ele se propõe a fazer na prática é possibilitar a execução de testes na rede e, depois de concluídos, exibir os resultados dos mesmos. Nos casos em que se deseje um monitoramento ao longo de um intervalo de tempo, o usuário deve fornecer valores para os parâmetros que indicam a freqüência e periodicidade desejadas para o teste em questão. De fato, a simplicidade desta interface pode ser encarada como um ponto positivo já que a interação do usuário com o sistema é facilitada. O que este sistema tem de especial é a forma como estes testes são executados, que não fica evidente para o usuário.

O diagrama de casos de uso da Figura 3.3 ilustra estas duas funcionalidades básicas.

3.4.2 Modelagem do Sistema

O sistema é dividido em quatro módulos principais correspondentes aos quatro tipos de agente. Além disto, há dois módulos adicionais com as classes que representam as entidades do sistema (objetos de negócio) e as entidades de testes. Há também o módulo para serialização de algumas destas entidades para o formato XML. Estes módulos são apresentados a seguir.

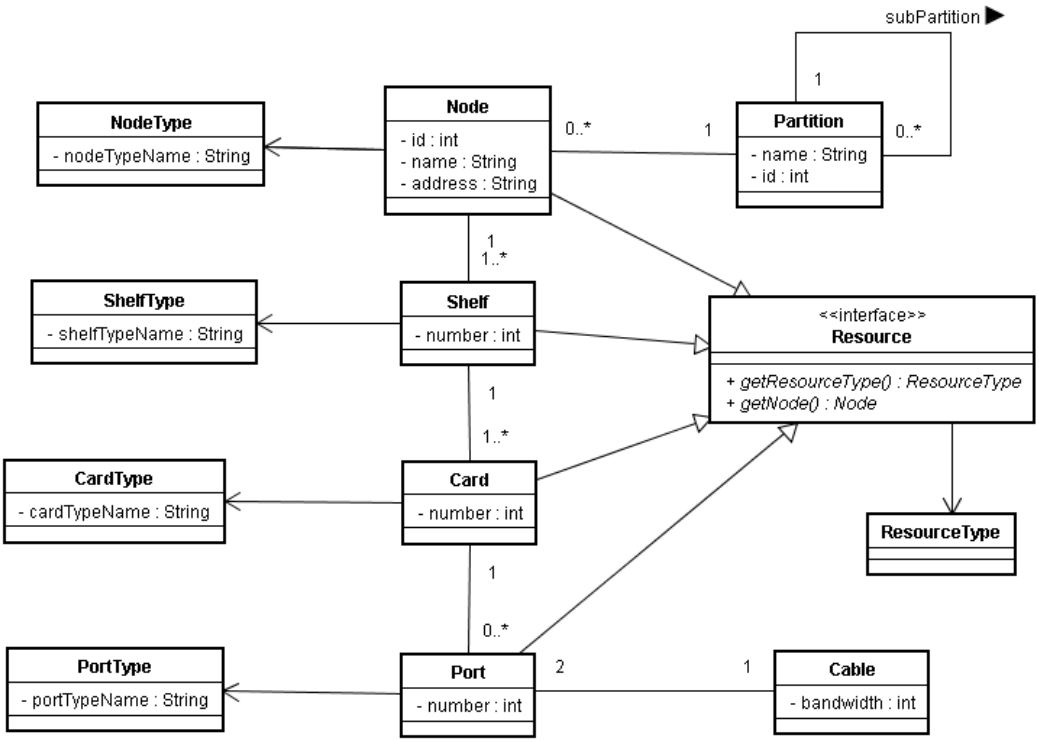

Figura 3.4: Diagrama de classes (simplificado) das entidades básicas do sistema

Módulo de Entidades

O módulo de entidades contém basicamente as classes utilizadas para representar recursos de rede sobre os quais o sistema irá executar os diversos testes. Os recursos de rede modelados são nós, *shelves*, cartões, portas e cabos. Também há classes para modelar os tipos de nós, *shelves*, cartões e portas existentes na rede. Há ainda uma classe para modelar as partições da rede.

As classes básicas podem ser visualizadas na Figura 3.4.

Módulo de Conversão de Entidades

O módulo de conversão de entidades é fortemente relacionado com o módulo de entidades. Nele encontram-se as classes responsáveis pelo mapeamento entre as instâncias de objetos que modelam recursos de rede e documentos XML equivalentes. A serialização de objetos que modelam recursos de rede é necessária para possibilitar a troca de mensagens entre os diversos agentes de software que compõem o sistema.

A classe `AbstractConverter` contém os mecanismos de conversão de objetos Java (gerados pela API JAXB [JAXB]) em XML e vice-versa. Cabe às demais classes implementar as conversões entre estas classes geradas automaticamente e as classes de entidades exibidas na Figura 3.4.

A Figura 3.5 apresenta as classes do módulo de conversão de entidades.

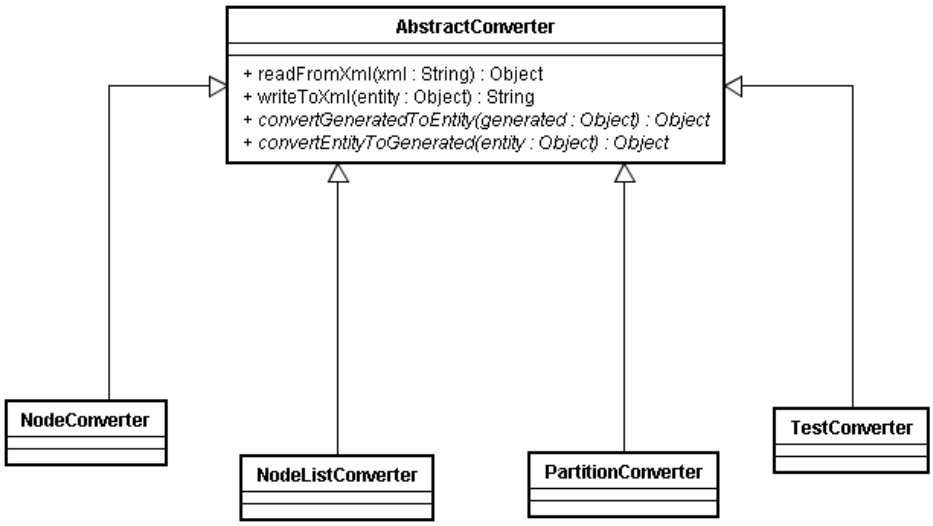

Figura 3.5: Diagrama de classes (simplificado) do módulo de conversão de entidades

Módulo de Testes

O módulo de teste reúne as classes relacionadas às abstrações de testes. A principal classe é uma classe abstrata (`Test`) da qual todos os testes devem herdar. Cada teste possui uma lista de parâmetros associada a si. Os parâmetros são os argumentos de entrada que o teste necessita para poder ser executado. Um parâmetro que virtualmente todos os testes devem ter, por exemplo, é o recurso de rede a que ele se aplica. Esta lista pode ter nenhum, um ou vários parâmetros dependendo do tipo do teste. Cada parâmetro tem um nome, uma descrição e, opcionalmente, um valor padrão.

Cada teste também possui um `TestCommand` associado, que é a classe em que de fato se encontra o método com a lógica para a execução do teste. Esta estratégia é baseada no padrão de projeto Comando [Gamma95], e provê um maior desacoplamento entre os diversos tipos de teste e o agente de teste, uma vez que o agente só precisa conhecer a interface e não as suas diversas implementações. Esta classe deve fornecer um resultado com uma mensagem e um nível de gravidade. Há três níveis de gravidade possíveis: em ordem (ou OK), alerta ou erro.

Na Figura 3.6 podem-se visualizar as classes do módulo de testes. Há também nesta figura duas classes que representam exemplos de testes.

Módulo do Agente Central

Este módulo contém as principais classes relacionadas ao agente central. Neste módulo destacam-se a classe `CentralAgent` que implementa o agente, a classe `ProvideTopologyBehaviour` que implementa o comportamento de

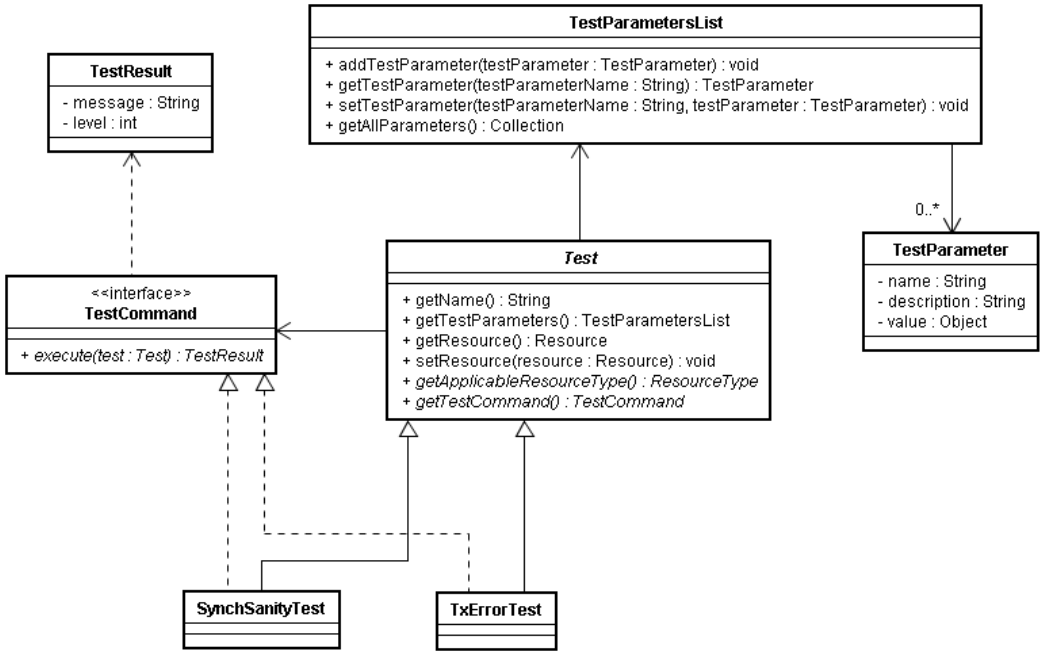

Figura 3.6: Diagrama de classes (simplificado) do módulo de testes

provimento de informações de topologia e as classes de acesso a dados (DAOs [Alur03]) que fazem a interface com o sistema centralizado de gerência.

A Figura 3.7 ilustra o módulo do agente central.

A interface `CentralDao` possui métodos para a extração de dados de topologia do sistema de gerência centralizado. Em última análise, toda e qualquer informação de topologia contida neste sistema de gerência e necessária para a execução de um teste será (indiretamente) obtida através de uma chamada a esta interface.

Módulo do Agente de Topologia

Este módulo contém as principais classes relacionadas ao agente de topologia. Neste módulo destacam-se se a classe `TopologyAgent` que implementa o agente de topologia e a classe `ProvideTopologyBehaviour` que implementa o comportamento de provimento de informações de topologia sobre os nós que gerencia. Este comportamento utiliza uma classe de acesso a dados (DAO) para recuperar as informações de topologia que precisa prover. Há ainda outros dois comportamentos: o `GetNodesFromPartitionBehaviour` que busca do agente central uma lista com os nós sobre os quais este agente é responsável, e o `CreateTestAgentBehaviour` que recebe solicitações de agentes de usuário para a criação de agentes de teste.

A Figura 3.8 ilustra o módulo do agente de topologia.

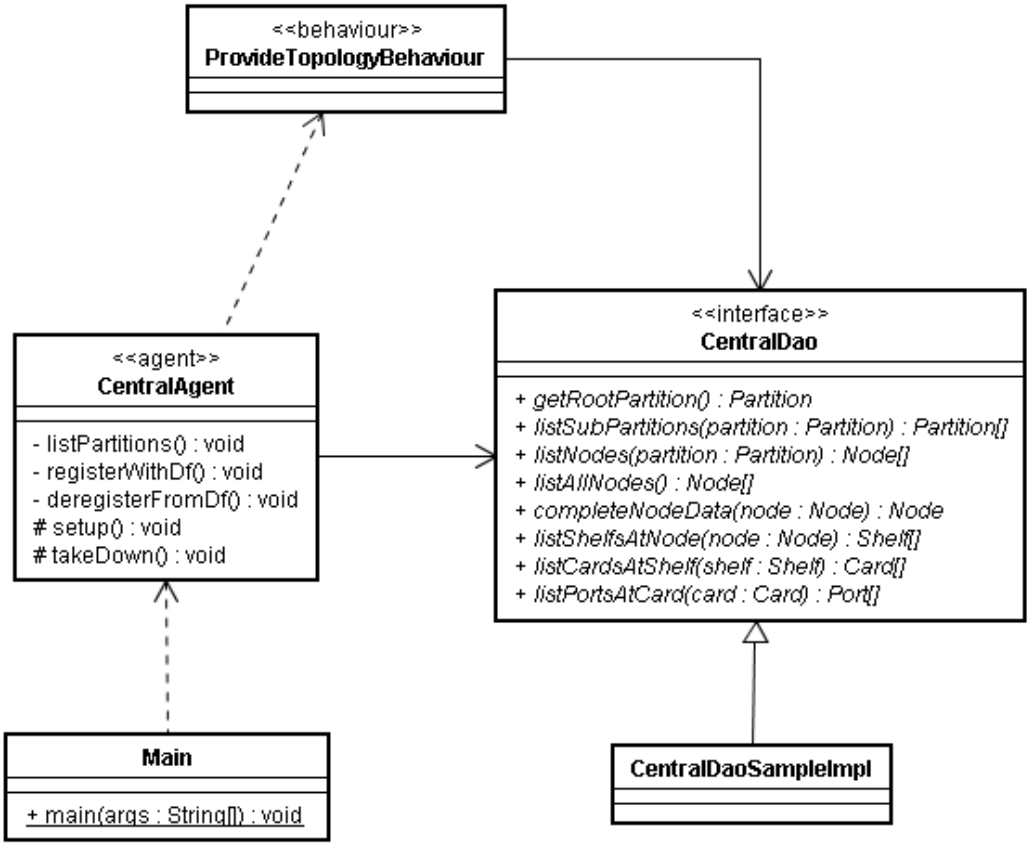

Figura 3.7: Diagrama de classes (simplificado) do módulo do agente central

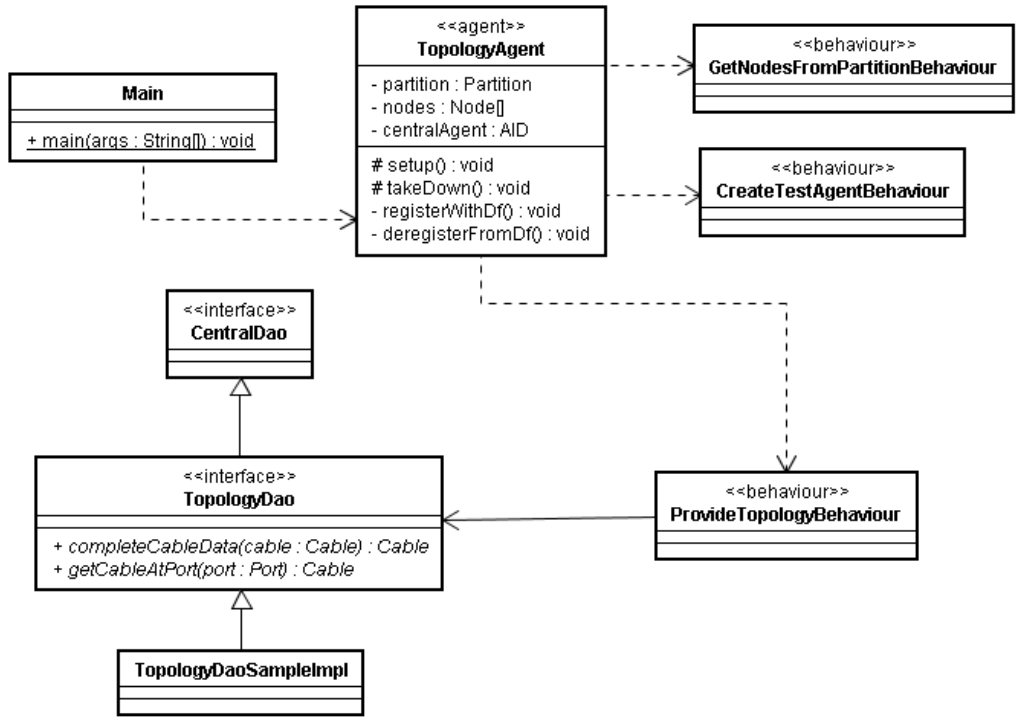

Figura 3.8: Diagrama de classes (simplificado) do módulo do agente de topologia

A interface `TopologyDao` possui métodos para a extração de dados de topologia mantidos pelo agente de topologia em uma base local que contém as informações que este obteve do agente central.

Há muitas semelhanças entre a interface `TopologyDao` exposta pelos agentes de topologia e a interface `CentralDao` exposta pelo agente central, uma vez que ambas são utilizadas para a obtenção de dados de topologia. A principal diferença entre as duas é que elas foram projetadas para atender diferentes clientes. A interface `CentralDao` deve ser utilizada primordialmente por agentes de topologia e a interface `TopologyDao` deve ser utilizada principalmente por agentes de teste. Em virtude disto, a interface `CentralDao` possui serviços mais simples para a recuperação de dados de topologia e inventário. Há duas razões principais para isto. A primeira razão é que não é aconselhável delegar tarefas complexas ao agente central uma vez que ele é o ponto mais crítico da arquitetura. Por isso as tarefas que ele desempenha devem ser tão simples quanto possíveis. A segunda razão é que é interessante fornecer aos agentes de testes serviços que facilitem a implementação dos mesmos no que diz respeito à disponibilização de informações de topologia e inventário de rede. Tais interfaces devem fornecer serviços não triviais como, por exemplo, identificar a porta de um nó remoto conectada (através de um cabo) a uma porta qualquer em questão. Esta relação entre as duas interfaces de topologia é evidenciada pelo fato de que a interface `TopologyDao` herda da interface `CentralDao`.

Módulo do Agente de Teste

Este módulo contém as principais classes relacionadas ao agente de teste. A principal classe deste módulo é a `TestAgent` que implementa o agente de teste. Há três comportamentos desempenhados por este agente: `ReceiveTestRequestBehaviour`, que recebe dos agentes de usuário as solicitações para a execução de testes, `ExecuteTestBehaviour`, que é quem de fato executa os testes, e `AskForTopologyBehaviour`, que é o comportamento responsável por solicitar informações de topologia ao agente de topologia correspondente.

Há ainda neste módulo a classe `TopologyDaoStubImpl`. Esta classe implementa a interface `TopologyDao` e é usada por testes que necessitem de informações de topologia para a sua execução. Esta classe transforma requisições de topologia em um documento (XML) que é utilizado pelo agente de teste para comunicar-se com um agente de topologia e solicitar do mesmo a informação de topologia desejada.

A Figura 3.9 ilustra as classes do módulo do agente de teste.

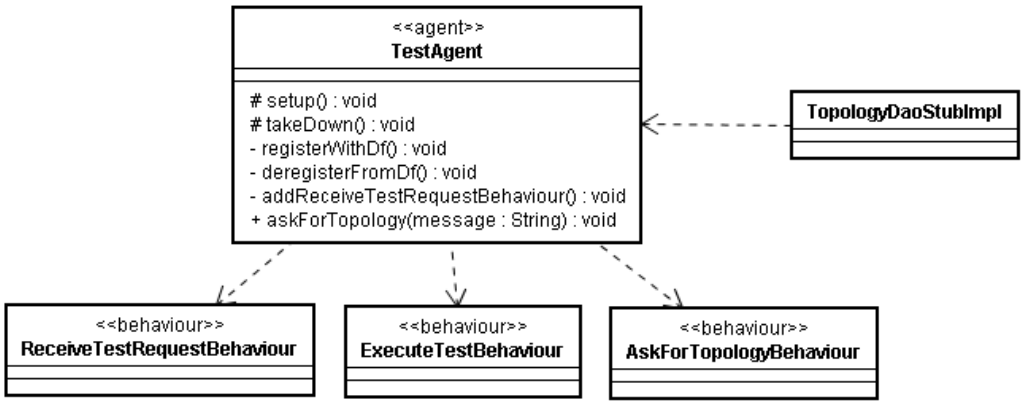

Figura 3.9: Diagrama de classes (simplificado) do módulo do agente de teste

Módulo do Agente de Usuário

Este módulo contém as principais classes relacionadas ao agente de usuário. A classe mais importante neste módulo é a classe `UserAgent` que implementa o agente de usuário. Ela utiliza as classes de comportamento `GetAllNodesBehaviour`, que busca uma lista com todos os nós da rede de um agente de topologia, `AskToCreateTestAgentBehaviour`, que solicita a um agente de topologia a criação de um agente de teste, e `AskToPerformTestBehaviour`, que solicita a um agente de teste a execução de um teste em um determinado recurso da rede. Este módulo ainda conta com algumas classes que representam as janelas da interface gráfica com o usuário.

A Figura 3.10 ilustra as classes do módulo do agente de usuário.

3.4.3 Comportamentos e Interações dos Agentes

Um aspecto interessante acerca da arquitetura do sistema diz respeito à dinâmica do mesmo, em particular aos comportamentos dos agentes de software e à interação entre diversos agentes. No contexto desta discussão, um *comportamento* de um agente representa uma tarefa que este deve desempenhar. Um comportamento pode ser uma atividade perene que deva ser executada indefinidamente (enquanto o agente permanecer ativo), como também pode ser uma atividade efêmera que deva ser executada e concluída rapidamente. Um agente ativo deve possuir pelo menos um comportamento, sendo também possível que ele possua mais de um simultaneamente[1].

A seguir são apresentados os principais comportamentos dos agentes do sistema proposto.

[1]Este conceito de comportamento é mapeado diretamente na classe Behaviour da plataforma JADE [JADE] usada neste projeto.

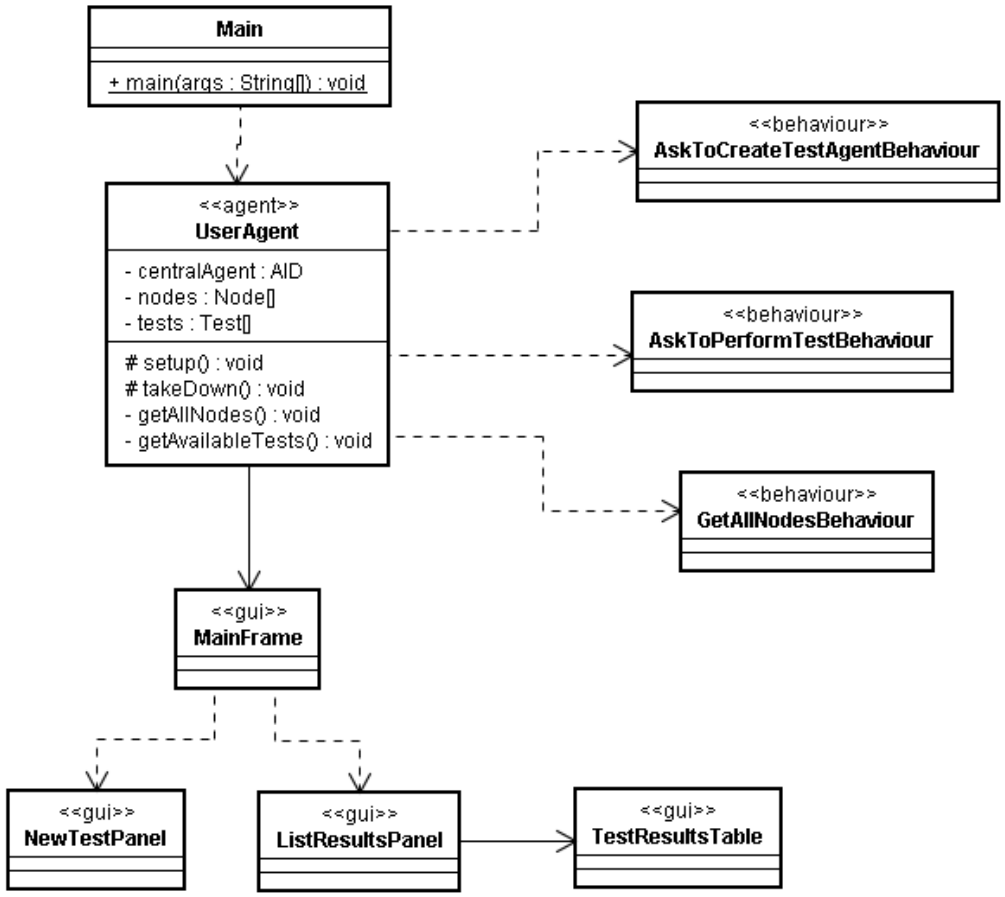

Figura 3.10: Diagrama de classes (simplificado) do módulo do agente de usuário

Agente Central

A principal atribuição do agente central é prover informações de topologia da rede de telecomunicações aos agentes de topologia. Logo, quando este é criado, ele cria e configura o comportamento que provê estas informações aos demais agentes. Em seguida, ele registra seus serviços no agente DF (*Directory Facilitator*), que provê um serviço de "páginas amarelas". O agente DF é definido no padrão [FIPA00023] e permite que um agente descubra outros agentes que ofereçam serviços necessários para que ele alcance algum de seus objetivos.

A Figura 3.11 ilustra o processo de criação do agente central.

Uma vez criado, o agente central se encontra pronto para responder a solicitações de informações de topologia. A principal função do comportamento `ProvideTopologyBehaviour` é prover acesso à interface `CentralDao`. Para que isto ocorra este comportamento aceita mensagens provenientes de agentes de topologia. Estas mensagens devem conter uma indicação de que tipo de informação desejam e do recurso de rede em questão (se for o caso). Estas indicações e os respectivos recursos são diretamente mapeados para métodos e

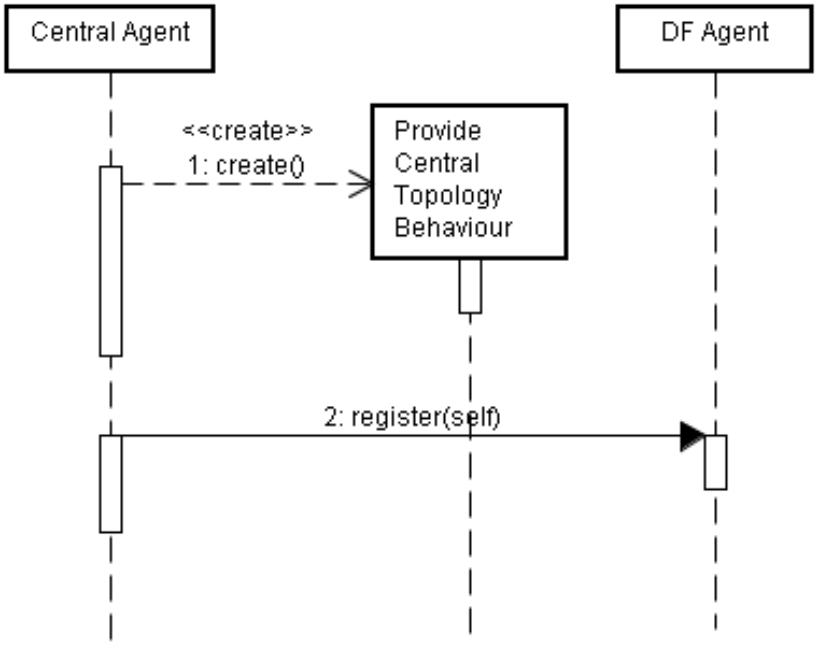

Figura 3.11: Diagrama de seqüência da criação do agente central

parâmetros da interface `CentralDao`. Caso não haja nenhum problema, o agente central enviará uma mensagem de resposta ao agente de topologia contendo as informações solicitadas.

Agente de Topologia

A principal atribuição do agente de topologia é prover informações de topologia aos demais agentes do sistema. Ele deve executar mecanismos para se manter atualizado com a topologia obtida do agente central. Além disso, o agente de topologia é o responsável pela criação de agentes de teste vinculados a equipamentos localizados na partição pela qual é responsável.

A primeira ação do agente de topologia quando de sua criação é buscar no agente DF a identificação do agente central. De posse desta informação o agente de topologia requisita ao agente central uma lista com os nós em sua partição da rede. Em seguida, o agente cria e configura o comportamento de prover informações de topologia para outros agentes do sistema e também o de criar agentes de teste quando a execução de um teste em sua partição for solicitada. Por fim o agente de topologia registra seus serviços no DF para que ele possa ser posteriormente contatado.

O processo de criação do agente de topologia é ilustrado na Figura 3.12.

Uma vez criado, o agente de topologia se encontra pronto para responder a solicitações de informações de topologia de nós pertencentes à partição sobre a qual é responsável. A principal função do comportamento

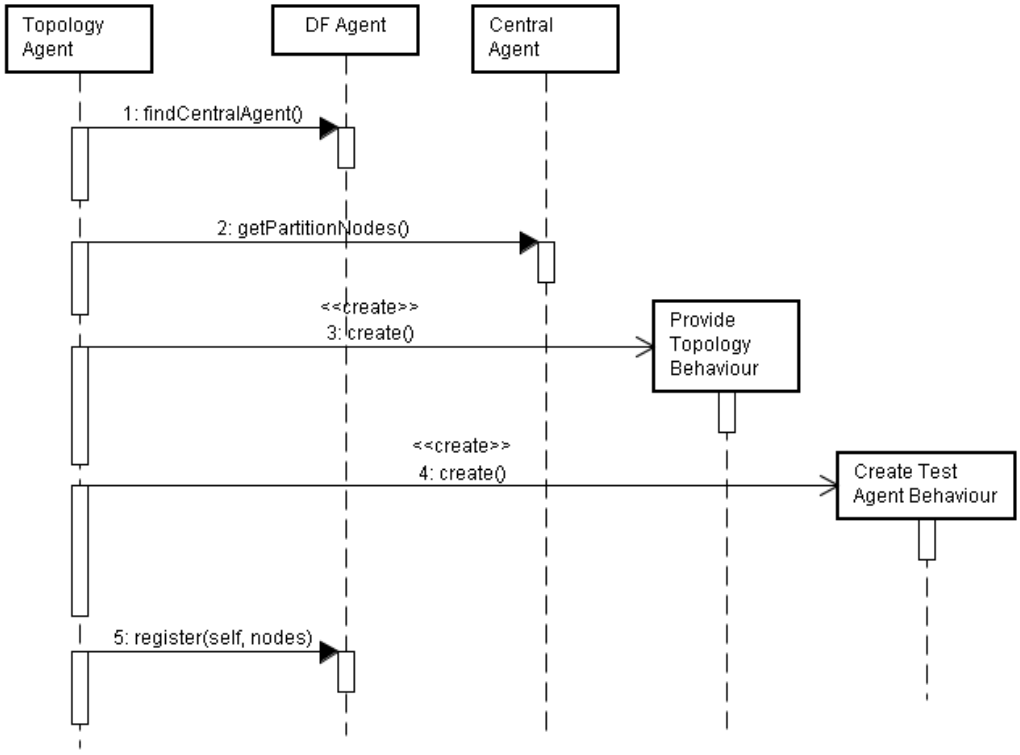

Figura 3.12: Diagrama de seqüência da criação de um agente de topologia

`ProvideTopologyBehaviour` é prover acesso à interface `TopologyDao`. Este comportamento é bastante semelhante ao comportamento que permite a troca de informações entre o agente central e agentes de topologia. Para que ocorra a troca de informações entre agentes de topologia e agentes de teste, este comportamento aceita mensagens provenientes destes agentes. Estas mensagens devem conter uma indicação de que tipo de informação desejam e do recurso de rede em questão (se for o caso). Estas indicações e os respectivos recursos são diretamente mapeados para métodos e parâmetros da interface `TopologyDao`. Caso não haja nenhum problema, o agente de topologia enviará uma mensagem de resposta ao agente de teste contendo as informações solicitadas. Para fornecer a resposta, o agente de topologia pode consultar a base que mantém localmente ou consultar outro agente de topologia ou mesmo o agente central.

Outro importante comportamento do agente de topologia é o que cria agentes de testes. Este comportamento espera por requisições de outros agentes, em particular os agentes de usuário, para a criação de um agente de teste. Este novo agente de teste a ser criado será necessariamente responsável por um nó que pertença à partição controlada pelo agente de topologia em questão.

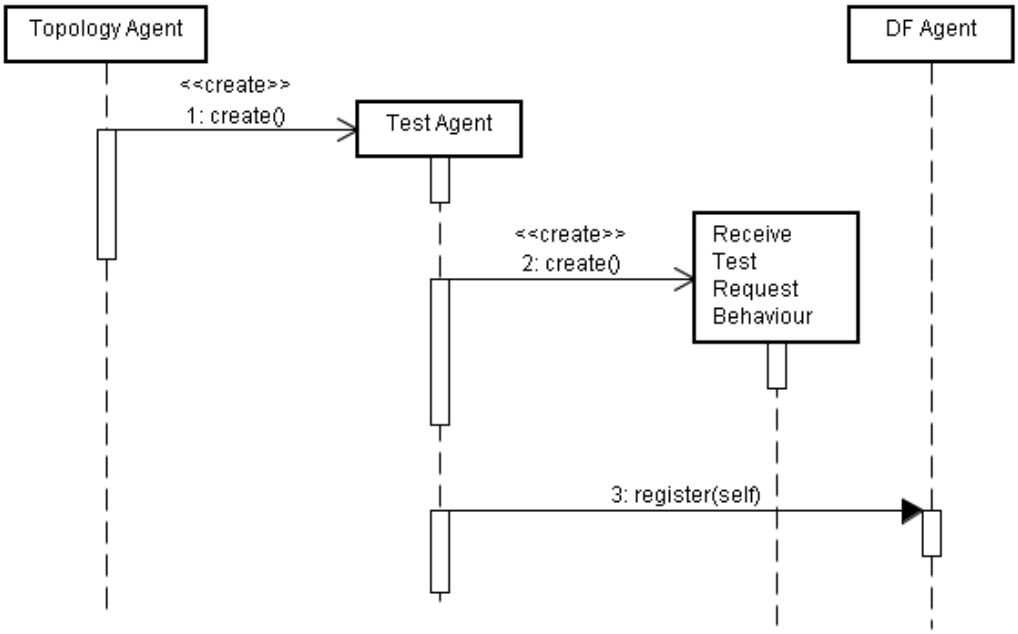

Figura 3.13: Diagrama de seqüência da criação de um agente de teste

Agente de Teste

A principal atribuição do agente de teste é realizar testes nos equipamentos. Um tipo comum de teste é a captura de estatísticas de uso do equipamento e a comparação com algum valor limite ou com uma série histórica. Para disparar a execução do teste ele invoca o método `execute` (da interface `TestCommand`) correspondente ao teste em questão.

Na arquitetura NeMaSA, os agentes de teste são criados por um agente de topologia quando um teste em um equipamento é solicitado e não há nenhum agente de teste responsável pelo mesmo. Para simplificar o projeto foi estabelecido que cada agente de teste fosse responsável por apenas um nó. Isto facilita o estabelecimento de uma hierarquia entre um agente de topologia e os agentes de testes responsáveis por nós pertencentes à partição controlada pelo agente de topologia. Esta hierarquia é interessante, pois permite que o agente de topologia efetue ações sobre agentes de testes em casos de mudanças na topologia da rede.

Quando um agente de topologia cria um agente de teste, este cria e adiciona um comportamento para receber as solicitações de execução de testes e registra seu serviço junto ao agente DF.

Este processo pode ver visualizado na Figura 3.13.

Uma vez criado o agente de teste, este fica responsável pela execução de testes relativos a um nó. Para que a execução de um teste seja solicitada a este agente, o comportamento `ReceiveTestRequestBehaviour` é utilizado. A função básica deste comportamento é receber mensagens com requisições de execução

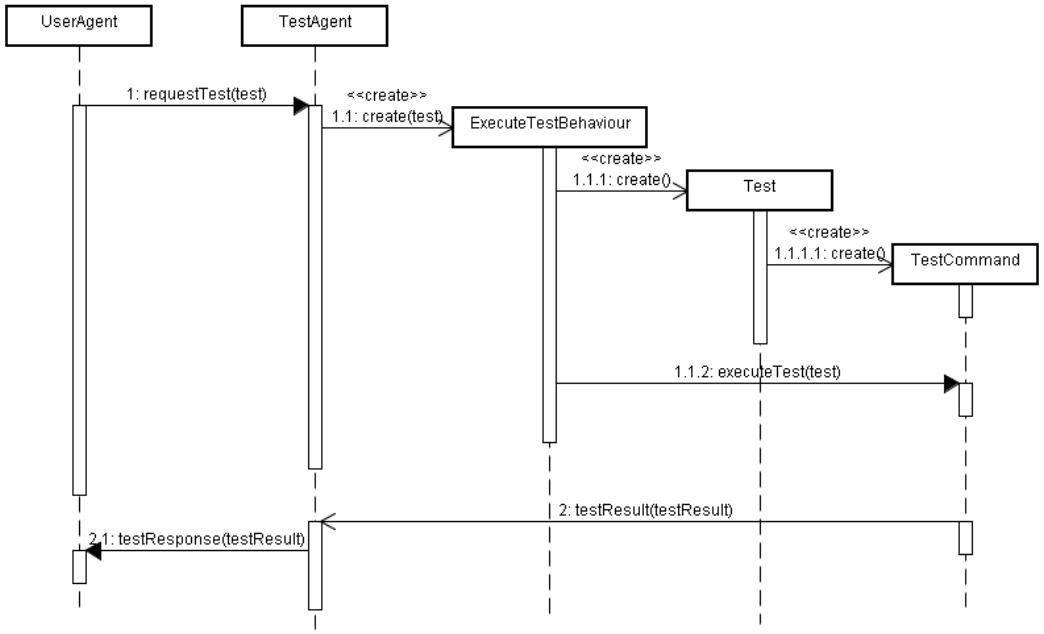

Figura 3.14: Processo de solicitação e execução de um teste

de testes de outros agentes (particularmente de agentes de usuários) e disparar o comportamento de execução do teste. Estas mensagens contêm a indicação de qual teste deve ser executado além dos parâmetros necessários para a execução do mesmo.

Este processo pode ser visualizado na Figura 3.14.

O mais importante comportamento do agente de teste é o comportamento que de fato executa o teste. Apesar de importante, este comportamento é bem simples, uma vez que a complexidade da execução do teste encontra-se encapsulada nas classes de testes. Uma vez encerrado o teste, o agente envia uma mensagem com os resultados do mesmo ao agente que solicitou a execução do teste. Este processo pode ser visualizado na Figura 3.14.

Um aspecto interessante, porém, é que o teste em execução pode eventualmente necessitar de alguma informação de topologia. Para que esta informação seja obtida do agente de topologia correspondente, o teste invoca o método da classe `TopologyDaoStubImpl` correspondente. Esta classe implementa a interface `TopologyDao` e dispara a comunicação entre o agente de teste e um agente de topologia de forma transparente para o teste em execução. Nesta classe a requisição de topologia é convertida em um documento XML que é passado para um método da classe do agente de teste. Neste método, o comportamento `AskForTopologyBehaviour` é criado e executado. Na execução deste comportamento, o agente de teste envia uma mensagem com esta requisição de informação de topologia ao agente de topologia que o criou. O agente de topologia obtém esta informação (de sua base local ou de algum outro agente) e

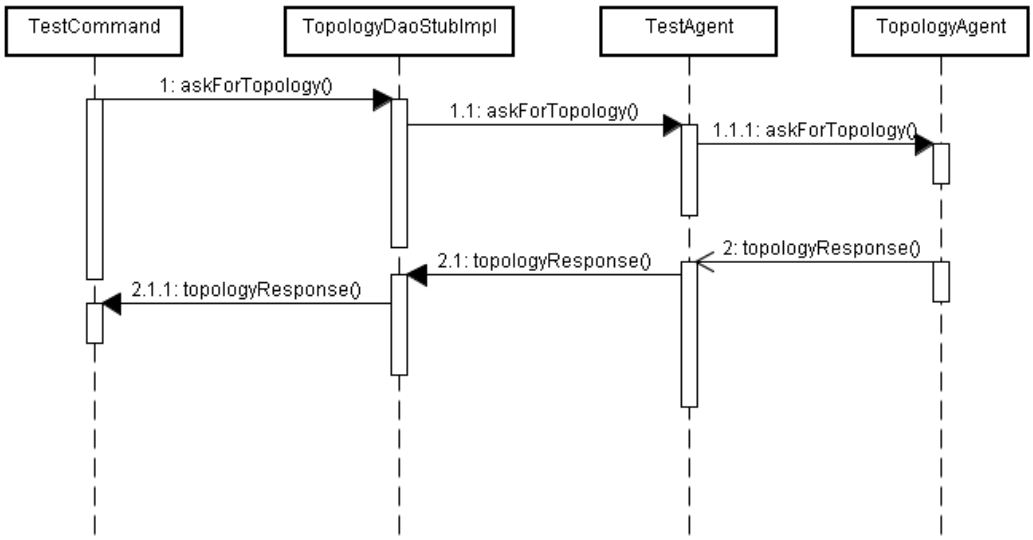

Figura 3.15: Processo de requisição de informações de topologia

envia a resposta ao agente de teste. Por fim, a informação de topologia é passada ao teste que a solicitou inicialmente, e este pode prosseguir.

A Figura 3.15 ilustra este processo.

Agente de Usuário

A principal atribuição do agente de usuário é desempenhar o papel de interface entre um usuário, geralmente um operador de rede, e o sistema. Para tanto, o agente de usuário cria e exibe janelas gráficas (GUI) para que o usuário entre com comandos ou diretivas a serem executadas pelo sistema. Esta interface gráfica também permite que o usuário visualize o resultado das ações do sistema na rede.

Quando um agente de usuário é criado, ele busca um agente de topologia junto ao agente DF para poder obter um inventário simples com todos os equipamentos da rede. Em seguida ele busca uma lista dos testes que o sistema é capaz de executar.

Uma representação deste processo é exibida na Figura 3.16. Estas informações são necessárias para a construção da interface gráfica apresentada na Figura 3.17.

Um dos eventos mais importantes do sistema é a solicitação da execução de um teste por parte do usuário. Quando isto ocorre, o agente de usuário busca junto ao agente DF o agente de teste responsável pelo nó escolhido. Caso não exista um agente de teste que possa atender esta demanda, o agente de topologia criará um novo agente de teste de acordo com o processo representado na Figura 3.13. Com o agente de teste já definido, o agente de usuário solicita a ele a execução do teste, que é conduzida através de um novo comportamento configurado pelo agente de teste.

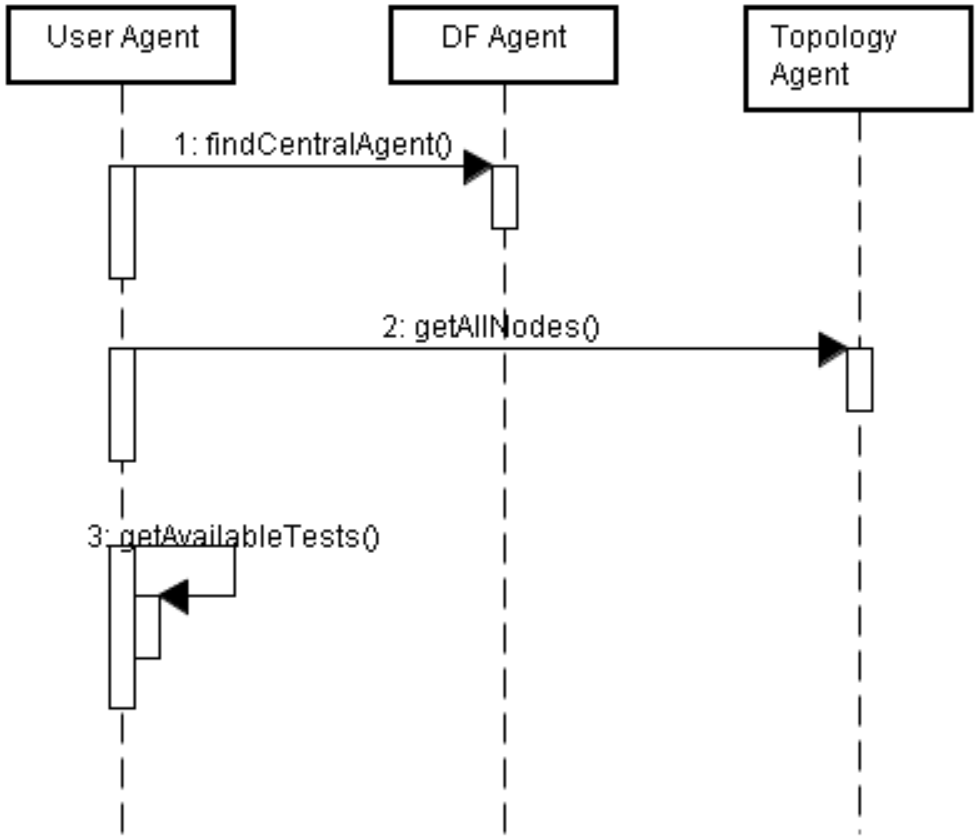

Figura 3.16: Diagrama de seqüência da criação do agente de usuário

Quando a execução do teste se encerra, o agente de teste envia ao agente de usuário o resultado do teste.

Este processo pode ser observado na Figura 3.18.

3.5 Aspectos de Implementação

O sistema é todo escrito em linguagem Java [Gosling05] e usa o *framework* JADE [JADE] como base para a implementação dos agentes de software. A troca de mensagens entre os agentes é feita através da linguagem ACL [FIPA00061]. Sempre que apropriado, as trocas de mensagens entre os agentes seguem os padrões de interação da FIPA, em particular o de consulta [FIPA00027] e o de requisição [FIPA00026].

Em muitas das mensagens trocadas entre os agentes, é necessária a transferência de entidades que representam recursos de rede, testes, resultados de testes, entre outros. No sistema estas entidades são modeladas como objetos Java, como foi discutido anteriormente. Nestes casos, estes objetos são serializados em formato XML, segundo o padrão JAXB [JAXB]. Para tanto é utilizada a biblioteca JaxMe 2 [JaxMe] da *Apache Software Foundation*. Esta estratégia tem a

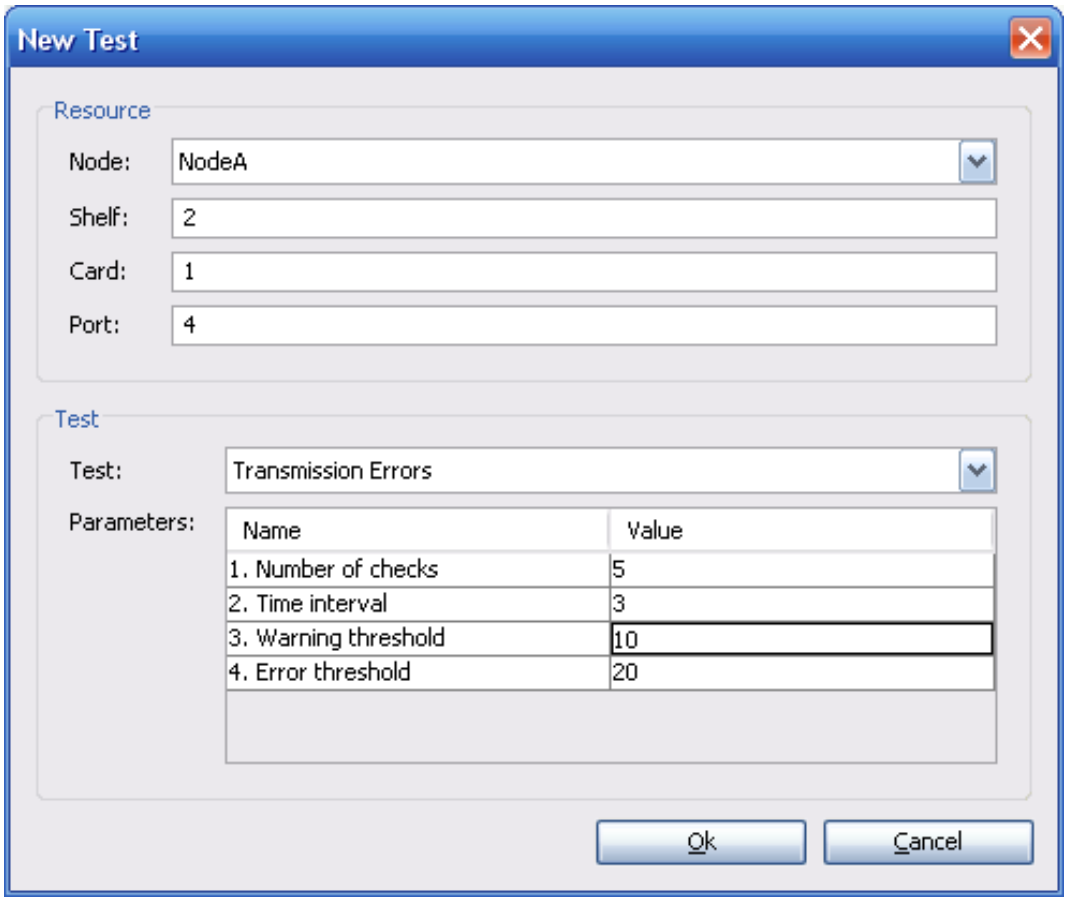

Figura 3.17: Janela para criação de um novo teste

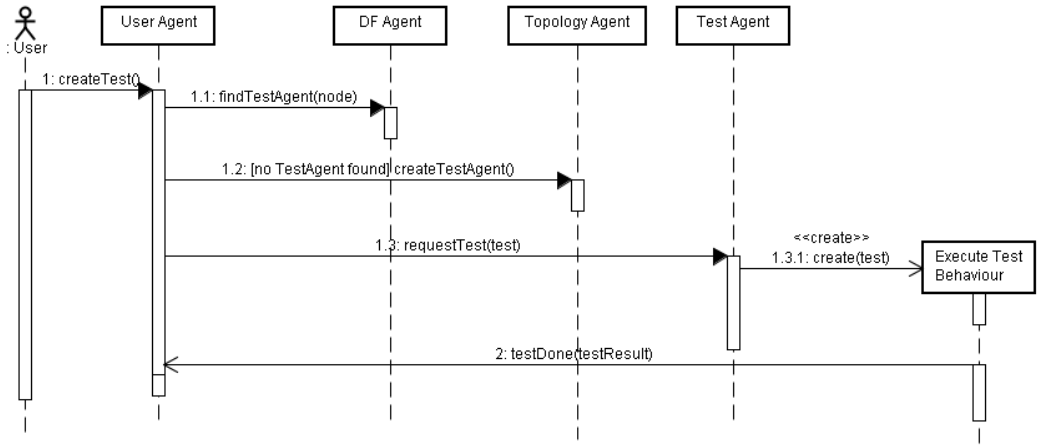

Figura 3.18: Diagrama de seqüência da criação de um teste

vantagem de não utilizar dados binários nas mensagens entre agentes, o que facilita a interoperabilidade com outros eventuais sistemas multi-agentes.

O sistema também utiliza o padrão IoC [Fowler04] para a configuração declarativa (e não programática) de objetos, baseado no *framework* Spring [Spring]. O uso deste padrão traz como vantagem um menor acoplamento entre os objetos do sistema. Este mecanismo é particularmente interessante para a criação do repositório de testes disponíveis aos agentes de testes. Para que um agente obtenha a referência de um teste qualquer basta que este possua o nome do mesmo e solicite a sua instanciação ao objeto que mantém o contexto do Spring. Todavia, um problema experimentado neste projeto e que limitou o uso desta técnica de IoC foi que os objetos que representam os agentes não podem ser configurados diretamente via Spring, pois eles possuem um ciclo de vida especial e devem ser gerenciados diretamente apenas pelo container de agentes do JADE.

4
Estudo de Caso

Com o propósito de melhor apresentar a arquitetura NeMaSA, assim como melhor ilustrar seu funcionamento, dois exemplos práticos de testes desenvolvidos sobre a arquitetura proposta serão apresentados neste capítulo. O primeiro teste é bem simples no que diz respeito à sua funcionalidade, mas ilustra bem a característica descentralizada da solução proposta. Além disto, este primeiro teste representa um tipo de teste comumente aplicado em redes reais devido à sua simplicidade. Por conta disto, sua apresentação é relevante para a presente discussão. O segundo teste é bem mais complexo, e ilustra a flexibilidade da solução proposta em cenários em que informações de topologia de rede são necessárias para o diagnóstico de uma falha.

4.1
Coleta de Estatísticas

Uma das atividades mais comuns no contexto de diagnóstico de falhas em redes de telecomunicações é o monitoramento de estatísticas coletadas por equipamentos de redes e mantidas nos mesmos. É comum haver em tais equipamentos contadores (ou acumuladores) para cada interface de rede (porta). Tais contadores mantêm diversos tipos de métricas tais como o número de *bytes* e pacotes trafegados, o número de erros detectados, o número de pacotes descartados, entre outros. Em alguns casos também é possível obter dados não diretamente relacionados ao tráfego de dados, mas sim relacionados ao funcionamento do equipamento propriamente dito, como por exemplo uso de CPU e memória. Tais informações podem ser utilizadas para a identificação de falhas ou outras situações anormais na rede. Um simples exemplo seria o monitoramento de erros detectados em uma interface de rede. Neste caso, uma taxa de erros acima de um determinado limite pode ser evidência de um problema grave tal como ruído na linha.

O primeiro teste selecionado para demonstrar o funcionamento da arquitetura NeMaSA é um teste de monitoramento de estatísticas de interfaces de rede. Tais estatísticas são diretamente coletadas nos equipamentos e comparadas a um valor limite escolhido pelo usuário. Caso o limite estabelecido seja ultrapassado, o teste pode executar um procedimento qualquer como o envio de um alerta aos operadores

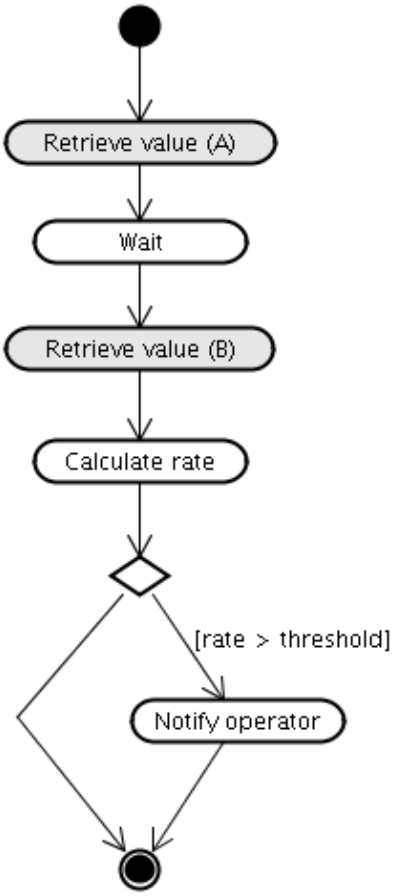

Figura 4.1: Representação gráfica do teste de erros de transmissão

de rede ou mesmo alguma ação corretiva como uma mudança de configuração do equipamento, por exemplo. A discussão seguinte ilustra como este teste pode ser modelado e implementado com a arquitetura NeMaSA.

Este simples tipo de teste necessita de apenas um tipo de tarefa. Esta tarefa é responsável por buscar no equipamento em questão o valor do contador que mantém a métrica de interesse. Neste caso, a métrica de interesse é o número de erros detectados em uma interface de rede (ou porta). Como esta tarefa não altera o estado do equipamento, ela pode ser classificada como um sensor. A lógica do teste é bem direta. O valor do contador é obtido em dois momentos distintos, separados por um intervalo de tempo pré-determinado. A taxa de erros é calculada e, caso seja maior que o limite definido pelo usuário, o teste termina com um diagnóstico de erro. Caso contrário, o diagnóstico indica uma situação normal. A Figura 4.1 ilustra graficamente a lógica do teste. Os elementos em tom escuro indicam a tarefa nos dois momentos descritos acima.

Para implementar este teste na arquitetura proposta, é necessário que se desenvolva duas classes principais: uma para representar as meta-informações sobre o teste, e outra para representar o algoritmo de diagnóstico do mesmo. As

Tabela 4.1: Meta-informações do teste de erros de transmissão

Meta-informação	Valor
Nome	"Transmission Errors Test"
Tipo de recurso de rede	Porta
Parâmetros de entrada	Valor limite (*threshold*) da taxa de erros; e porta (parâmetro implícito)

meta-informações necessárias são (i) o nome do teste, (ii) o tipo de recurso de rede sobre o qual este teste pode ser aplicado e (iii) o conjunto de parâmetros de entrada necessários para a sua execução. O parâmetro de entrada que define o recurso de rede a ser testado tem um tratamento especial e, por isso, não precisa ser declarado no conjunto de parâmetros de entrada do teste. De acordo com a API proposta, as meta-informações do teste devem ser definidas em uma classe que extenda a classe abstrata `Test`. O algoritmo do teste, por sua vez, deve ser representado em uma classe que implemente a interface `TestCommand`, definida pela API da arquitetura. Esta interface define apenas um método, que tem como parâmetro de entrada um `Test` e retorna um `TestResult`. Opcionalmente, pode-se construir uma única classe que extenda a classe `Test` e implemente a interface `TestCommand`, satisfazendo os dois requisitos acima apresentados.

As meta-informações do teste em questão estão expostas na Tabela 4.1. A Figura 4.2 exibe a listagem da classe `TransmissionErrorsTest` que representa o teste em questão. Esta classe contém tanto as meta-informações do teste quanto o seu algoritmo de diagnóstico.

Por fim, é também interessante discutir o comportamento da aplicação quando da execução deste teste. Em particular é interessante discutir o comportamento dos agentes e as trocas de mensagens entre os mesmos para a execução do teste em discussão. A execução deste teste se inicia com a solicitação por parte de um usuário do diagnóstico da taxa de erros de uma porta qualquer de sua escolha. Esta solicitação é feita através de uma interface gráfica ilustrada pela Figura 3.17. A partir desta solicitação o agente de usuário passa a agir no sistema em nome do usuário com o objetivo de que o teste seja executado. Sua primeira ação é solicitar ao agente DF um agente de teste que possa de fato executar o teste. Caso não haja um agente de testes disponível, o agente de usuário solicita ao agente de topologia responsável pela partição em que se encontra o nó a criação de um novo agente de teste. Uma vez definido o agente de teste responsável pela execução do teste em questão, o agente de usuário envia uma mensagem com a solicitação da execução do teste assim como os dados necessários, como as meta-informações do teste e os valores dos parâmetros de entrada. O agente de teste, então inicia o comportamento de execução de teste que chama o método `execute` da interface `TestCommand`

```
public class TransmissionErrorsTest extends Test implements TestCommand {

  public TransmissionErrorsTest() {
    super("Transmission Errors Test");
  }

  public ResourceType getApplicableResourceType() {
    return ResourceType.PORT;
  }

  protected TestParametersList initTestParametersList() {
    TestParametersList parameters = new TestParametersList();
    parameters.addTestParameter(new TestParameter("Error threshold"));
    return parameters;
  }

  public TestCommand getTestCommand() {
    return this;
  }

  public TestResult execute(Test test) {
    Port port = (Port) this.getResource();
    double threshold = Double.parseDouble(test.getTestParameters()
        .getTestParameter("Error threshold").getValue());

    RetrieveErrorCountFromPortTask task = new
        RetrieveErrorCountFromPortTask(port);

    double valueA = task.getErrorCount();

    try {
      Thread.sleep(10 * 1000);
    } catch (InterruptedException e) {
      throw new RuntimeException(e);
    }

    double valueB = task.getErrorCount();

    double rate = (valueB - valueA) / 10.0;

    TestResult testResult = new TestResult();
    if (rate > threshold) {
      testResult.setMessage("Transmission errors are over threshold");
      testResult.setLevel(Level.ERROR);
    } else {
      testResult.setMessage("Transmission errors are below threshold");
      testResult.setLevel(Level.OK);
    }

    return testResult;
  }
}
```

Figura 4.2: Listagem da classe `TransmissionErrorsTest`

descrita acima. Quando este método acaba, o objeto `TestResult` é transmitido de volta ao agente de usuário através de uma mensagem. O agente de usuário, de posse deste resultado, exibe ao usuário as mensagens resultantes do teste em uma interface gráfica. Este processo está visualmente representado na Figura 3.18.

Um aspecto interessante da arquitetura proposta é evidenciado pelo exemplo acima exposto. Este aspecto é o mecanismo distribuído para a execução de testes. Conforme pode ser depreendido da discussão acima, o processo de execução de um teste pode ser dividido entre diversas entidades que estejam fisicamente distribuídas pela rede. Esta distribuição pode ser presenciada neste exemplo de teste em pelo menos dois momentos distintos. Primeiramente, não há restrições de onde o usuário e o agente de usuário devam estar para interagir com o sistema. Em segundo lugar, o agente de teste pode estar fisicamente localizado em uma máquina próxima ao nó sendo testado. Esta proximidade permite uma menor latência nas comunicações entre o agente de teste e o equipamento, assim como contribui para um menor tráfego pela rede de dados resultantes de solicitações de tarefas que fazem parte de algoritmos de testes. O único momento em que há um comportamento centralizado é quando um agente de usuário solicita a um agente de topologia a criação de um agente de teste. Porém, uma vez definido o agente de teste responsável pela execução de testes em um nó, toda a ação do sistema para a execução de testes ocorre de forma independente das entidades centralizadas, a saber, o agente central (e o repositório centralizado que gerencia) e os agentes de topologia.

4.2 Fonte de Sincronismo

O segundo exemplo prático de teste discutido neste capítulo é significativamente mais complexo que o primeiro. Enquanto o primeiro exemplo de teste serviu para ilustrar o mecanismo básico para a implementação de testes na arquitetura proposta, este segundo exemplo de teste ilustra a dinâmica de um teste em que informações de topologia da rede são necessárias para o diagnóstico de uma falha na rede.

Este segundo exemplo prático de teste, que faz uso de informações de topologia, tem como objetivo verificar o sincronismo do relógio (*clock*) de um equipamento de rede. Muitos equipamentos de telecomunicações de maior porte devem ajustar seu relógio interno de acordo com alguma fonte de sincronismo externa para garantir seu correto funcionamento [Bregni02]. Tal fonte de sincronismo geralmente é um receptor de GPS [GPS]. Também é possível, entretanto, que o sincronismo seja obtido de um outro nó da rede que esteja devidamente sincronizado. Para tanto, este deve estar direta ou indiretamente (através de outros nós) conectado a um receptor de GPS, o que acaba formando uma

"cadeia de sincronismo", que começa no nó em questão e vai até o receptor de GPS, passando por nós intermediários. O exemplo mais comum de falha neste contexto é a perda de conectividade com a fonte de sincronismo. O diagnóstico desta situação é relativamente complexo, pois ele exige a identificação da cadeia de sincronismo do nó em questão e de todos os nós a ela pertencentes. O que torna este problema ainda mais interessante é que um nó dificilmente possui uma referência direta ao nó seguinte da cadeia de sincronismo. Geralmente há apenas uma referência à porta que liga os dois nós. Para que se descubra o outro nó, é preciso identificar o cabo conectado a esta porta e, só então, o nó na outra ponta do cabo. Este é, portanto, um exemplo de teste em que a informação de topologia é essencial para a obtenção do diagnóstico de falhas.

Há pelo menos três condições de anormalidade que se deseja detectar em relação ao sincronismo de relógios de equipamentos de telecomunicações. Para uma melhor compreensão deste problema, entretanto, uma breve explanação sobre o comportamento de tais equipamentos se faz necessária. Geralmente, equipamentos de telecomunicações de grande porte são projetados de forma a serem capazes de se manterem corretamente sincronizados por um certo intervalo de tempo mesmo quando a cadeia de sincronismo é rompida. Esta situação é denominada *Hold Over*, e não caracteriza uma falha uma vez que o sincronismo ainda é mantido. Entretanto, tal situação exige alguma forma de intervenção para que uma situação de normalidade seja restabelecida. Caso esta condição seja mantida indefinidamente, sem que a conexão com uma fonte confiável de sincronismo seja restabelecida, o tempo máximo de sincronismo garantido pelo nó se esgota e o mesmo parte para um estado denominado *Free Run*. Nós nesta condição estão, para efeitos práticos, dessincronizados, o que pode acarretar falhas no tráfego de dados entre nós adjacentes. Tal condição sinaliza uma falha grave na rede que deve ser rapidamente corrigida. Uma terceira condição de anormalidade ocorre quando há a presença de laços (*loops*) na cadeia de sincronismo. Isto ocorre, por exemplo, se um nó A tem um nó B como sua fonte de sincronismo, o nó B depende de um nó C e, por fim, o nó C obtém de A seu sincronismo. Em casos como este, todos os nós podem ser considerados dessincronizados apesar de não estarem conscientemente no modo *Free Run*.

O algoritmo para a detecção de tal falha precisa percorrer a cadeia de sincronismo desde o nó original até encontrar uma fonte confiável, um nó em *Hold Over* ou um nó em *Free Run*. Estes casos indicariam, respectivamente, uma situação normal, um alerta e um erro. Além disto, este algoritmo também precisa manter uma lista de todos os nós da cadeia percorridos até então para poder detectar a existência de laços. Caso algum nó seja percorrido mais de uma vez, caracteriza-se a existência de um laço, que é um erro grave. A Figura 4.3 ilustra graficamente este

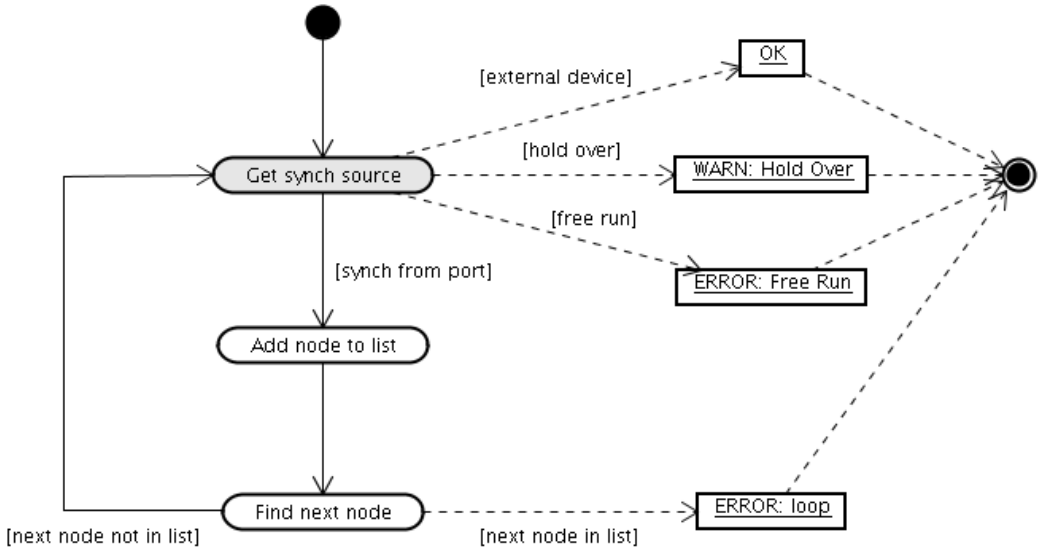

Figura 4.3: Representação gráfica do teste de sincronismo

algoritmo. Há uma tarefa (em tom escuro) que busca no equipamento a sua atual fonte de sincronismo. O resultado desta tarefa pode ser (i) sincronismo proveniente de dispositivo externo (GPS), (ii) nó em modo *Hold Over*, (iii) nó em modo *Free Run*, ou (iv) sincronismo proveniente de outro nó. Neste último caso, há a indicação da porta por onde vem o sincronismo. Sempre que esta quarta situação for encontrada, é necessário se descobrir o nó remoto ligado à porta em questão. Este é o nó seguinte na cadeia de sincronismo. Para que tal informação seja obtida, deve-se consultar uma base com a topologia da rede.

A Figura 4.4 exibe a listagem de como tal algoritmo pode ser implementado com a API fornecida pela arquitetura. A organização básica desta classe segue as mesmas características da classe do teste simples discutida anteriormente. Há algumas diferenças entre ambas, entretanto, e alguns comentários adicionais sobre esta implementação fazem-se necessários. Primeiramente, o tipo de recurso sobre o qual este teste atua é diferente. No caso anterior, o teste atuava sobre portas. Este atua sobre nós. Em segundo lugar, este teste não espera nenhum parâmetro de entrada além do recurso sobre o qual atua (nó). Por isto, ele retorna uma lista de parâmetros de entrada vazia. Em terceiro lugar, como este teste possui uma característica cíclica em seu comportamento, o algoritmo do teste é implementado através de chamadas recursivas ao método privado `executeTestOnNode`, que é invocado para cada nó pertencente à cadeia de sincronismo. Por fim, a característica determinante deste teste, que é a necessidade de consultas a informações de topologia é evidenciada pelo uso da interface `TopologyDao`. Uma implementação desta interface é passada pelo agente de teste para o objeto que implementa o teste. Nesta implementação, o agente de teste cria uma mensagem baseada no método invocado pelo teste assim como os parâmetros do método. Esta mensagem é enviada

```
public class SynchSanityTest extends Test implements TestCommand {

  private TopologyDao topologyDao;

  private Set<Node> visitedNodes;

  public SynchSanityTest() {
    super("Synchronization Sanity");
  }

  public void setTopologyDao(TopologyDao topologyDao) {
    this.topologyDao = topologyDao;
  }

  public ResourceType getApplicableResourceType() {
    return ResourceType.NODE;
  }

  protected TestParametersList initTestParametersList() {
    return new TestParametersList();
  }

  public TestCommand getTestCommand() {
    return this;
  }

  public TestResult execute(Test test) {
    Node node = (Node) test.getResource();
    visitedNodes = new HashSet<Node>();
    return executeTestOnNode(node);
  }

  private TestResult executeTestOnNode(Node node) {
    RetrieveSynchSourceTask task = new RetrieveSynchSourceTask(node);
    SynchSourceType synchSourceType = task.getSynchSourceType();

    switch (synchSourceType) {
    case EXTERNAL:
      return new TestResult("Node is correctly synchronized", Level.OK);

    case HOLD_OVER:
      return new TestResult("Node is in Hold Over mode", Level.WARNING);

    case FREE_RUN:
      return new TestResult("Node is in Free Run mode", Level.ERROR);

    case PORT:
      visitedNodes.add(node);

      Port localPort = task.getSynchSourcePort();
      Cable cable = topologyDao.getCableAtPort(localPort);
      Port remotePort = cable.getEndA();
      if (remotePort.equals(localPort)) {
        remotePort = cable.getEndB();
      }
      Node remoteNode = remotePort.getNode();

      if (visitedNodes.contains(remoteNode)) {
        return new TestResult("Loop found at synchronization chain",
            Level.ERROR);
      } else {
        return executeTestOnNode(remoteNode);
      }

    default:
      throw new RuntimeException("This should never happen");
    }
  }
}
```

Figura 4.4: Listagem da classe `SynchSanityTest`

pelo agente de teste para o agente de topologia correspondente. Assim que o agente de teste recebe a resposta do agente de topologia, esta mensagem é interpretada e passada de volta à execução do teste com o retorno do método invocado pelo teste. Este processo é ilustrado na Figura 3.15.

Como pode ser observado nos dois exemplos de testes apresentados neste capítulo, a arquitetura proposta permite que a implementação de testes seja bem simples, mesmo quando informações de topologia são necessárias para o diagnóstico de falhas. Há, entretanto, um problema com a forma com que o segundo teste foi implementado. Uma das vantagens da arquitetura proposta levantada na discussão do primeiro exemplo de teste foi que havia uma proximidade geográfica entre o agente de teste e os nós sobre os quais ele atuava. Da forma como este segundo teste foi implementado, esta proximidade geográfica não pode mais ser garantida, uma vez que o teste executa tarefas sobre todos os nós pertencentes à cadeia de sincronismo do nó sob diagnóstico, e não há garantias que estes nós estejam geograficamente próximos. Na prática esta proximidade é esperada, visto que não é comum criar cadeias de sincronismo de relógios com equipamentos muito distantes entre si. De qualquer maneira, a solução da arquitetura em seu estágio atual não é a melhor possível. Uma alternativa mais interessante seria deslocar a execução do teste para outro agente de teste mais próximo dos nós, mais este mecanismo não foi implementado na presente versão da arquitetura.

5
Trabalhos Relacionados

5.1
Introdução

A área de telecomunicações, em especial a área de gerência de redes, representa um dos mais promissores e desafiadores domínios para a aplicação de tecnologias de agentes de software [Magedanz96, Muller97]. Tal fato é evidenciado pela quantidade de trabalhos encontrados na literatura que exploram o uso de agentes de software no domínio de telecomunicações. Exemplos destes trabalhos podem ser encontrados em diversas publicações tais como anais de eventos acadêmicos [Albayrak98a, Albayrak98b, Albayrak99], livros [Hayzelden99a, Hayzelden01] e artigos [Cheikhrouhou98, Hayzelden00].

Há pelo menos quatro importantes razões para este comum casamento entre agentes de software e sistemas de telecomunicações. Em primeiro lugar, uma rede de telecomunicações é um ambiente naturalmente distribuído, composto por diversos elementos (ou nós) relativamente independentes entre si. A modularidade intrínseca a SMA (devido ao fato de serem compostos por agentes relativamente independentes) se encaixa bem neste contexto. Um segundo fator é o fato de haver por parte dos operadores de rede um grande desejo pela automatização de processos, com vistas a reduções de custos de operação da rede. As características de autonomia e pró-atividade comuns a agentes de software vão ao encontro deste anseio por automatização. Um terceiro fator para a aplicação de agentes de software é a complexidade comum a ambientes de redes em que se destaca a heterogeneidade de equipamentos, protocolos e fabricantes que tipicamente compõem as redes de telecomunicações. A aplicação do paradigma de sistemas multi-agentes traz uma grande flexibilidade para a implementação de novas funcionalidades de controle e gerência da rede. Por fim, um fator que muitos autores argumentam em favor do uso de agentes de software (e que não é específico ao domínio de telecomunicações) é que a tecnologia de agentes de software provê um nível de abstração mais apropriado que metodologias mais tradicionais (como a de orientação a objetos) para a modelagem e o projeto de sistemas complexos.

A partir de meados da década de 90, a comunidade acadêmica vem pesquisando intensamente a aplicação de agentes de software em redes de telecomunicações. As aplicações têm sido bastante diversificadas. Há trabalhos como [Minar99, Luckschandl01, González02] que sugerem o uso de algoritmos baseados em agentes de software para determinar rotas em redes de telecomunicações. O projeto IMPACT [Hansen01], por sua vez, utiliza agentes de software para controle de redes ATM [I.321]. Através destes agentes, é possível desempenhar tarefas de controle de admissão de conexão (CAC), roteamento e gerência de circuitos virtuais (VCs). Para tanto, estes agentes se comunicam com *switches* através de protocolos de sinalização ATM.

Agentes de software também têm sido usados para garantia de qualidade de serviço (QoS) em redes de telecomunicações. Em [DeMeer00], por exemplo, é proposto um mecanismo para gerência de QoS em redes IP que é sensível a variações nos requisitos das aplicações de usuário assim como variações no estado da rede. Os agentes são capazes de decidir dinamicamente a quantidade de recursos (como largura de banda) que será reservada para cada *link* virtual que liga um par de usuários. Para a reserva de recursos, os agentes usam mecanismos de sinalização RSVP [RFC2205]. Já em [Ye02], se propõe um mecanismo baseado em agentes de software para garantia de QoS em redes sem fio. O trabalho considera a questão do usuário em movimento e algumas das implicações desta questão no que diz respeito ao uso da rede. Neste trabalho são expostas duas abordagens distintas para a garantia de qualidade de serviço nestas condições. A primeira é pró-ativa e prega a reserva antecipada de banda na célula com a maior probabilidade de ser visitada pelo usuário em seguida. A segunda é reativa e busca gerenciar requisições de banda assim que elas ocorrem. Por fim, estas duas abordagens são integradas em uma estratégia única.

Outra vertente de trabalhos que aplicam agentes de software no domínio de telecomunicações explora a capacidade de negociação entre agentes com a finalidade de oferecer a usuários da rede serviços de melhor qualidade ou com a melhor razão entre custo e benefício possível. Um exemplo é o encontrado em [FIPA00082]. Neste documento, é proposta uma arquitetura em que agentes representantes de usuários negociam com agentes representantes de provedores de serviços o provisionamento de serviços de VPN [Venkateswaran01]. Estes agentes, por sua vez, negociam com agentes representantes dos administradores de redes de telecomunicações a utilização dos recursos de rede necessários para a prestação do serviço de VPN. O objetivo do agente de usuário é ser capaz de identificar o provedor de serviços que lhe ofereça as melhores condições de qualidade e preço. Outro trabalho em que se explora a negociação entre agentes é [Gibney01]. Nele, é proposto um sistema em que agentes de software

negociam entre si o estabelecimento de circuitos (ou chamadas) em uma rede de telecomunicações. Ao contrário da arquitetura apresentada em [FIPA00082], os agentes de usuários propostos por Gibney *et al.* (aqui chamados de agentes de chamada) não negociam com agentes representantes de organizações comerciais (como provedores de serviço e administradores de redes), mas sim com agentes que representam entidades pertencentes à rede, como elos (*links*) e circuitos (*paths*). O objetivo deste sistema é racionalizar o uso dos recursos da rede de modo a evitar que nela ocorram congestionamentos.

Uma linha comum de aplicação de agentes de software para gerência de redes é através dos chamados agentes móveis [Morreale98, Bieszczad98]. Agentes móveis são agentes de software que se movem de um nó para outro da rede a seu próprio critério, levando consigo seu estado e código executável para o novo hospedeiro. A linha de pesquisa de agentes móveis aplicados à gerência de redes obteve bastante popularidade a partir de meados da década de 1990 e há diversos trabalhos publicados na área. Alguns exemplos podem ser encontrados em [Karmouch99, Horlait00, Pierre01, Karmouch02, Horlait03].

O uso de agentes de software estáticos pode geralmente proporcionar uma solução suficiente para diversos problemas de gerência de redes, mas a aplicação de agentes móveis traz mais flexibilidade para as soluções e abre espaço para novas possibilidades e benefícios. A argumentação a favor do uso de agentes móveis normalmente explora a questão da interação local entre o agente de software e o recurso gerenciado. Esta interação local proporciona benefícios geralmente associados a questões de desempenho (tais como redução de tráfego na rede) e de extensibilidade (tais como a atualização e instalação de software nos equipamentos gerenciados). Também é comum a argumentação de que agentes móveis garantam mais confiabilidade e robusteza aos sistemas que os empregam [Bieszczad98]. Tal argumento é particularmente apropriado para redes com enlaces intermitentes ou pouco confiáveis nas quais soluções baseadas em trocas de mensagens têm sua eficácia bastante comprometida. Um ponto negativo relacionado à aplicação de agentes móveis é o acréscimo na complexidade de sua implementação e em aspectos relacionados à segurança. Estudos quantitativos sobre o desempenho de sistemas de gerência baseados em agentes móveis podem ser encontrados em [Rubinstein99, Zhu01].

Apesar da aplicação de agentes de software para a resolução de problemas em diversos contextos no domínio de telecomunicações, é na área de gerência de redes que a aplicação deste novo paradigma de engenharia de software parece ter encontrado um solo mais fértil para a pesquisa e desenvolvimento. Há diversos trabalhos nesta categoria. Há, por exemplo, os que exploram agentes móveis, os

que exploram monitoramento de elementos de rede e ainda alguns que propõem arquiteturas mais genéricas para gerência através de agentes.

Nas seções seguintes três trabalhos que apresentam características semelhantes à arquitetura proposta neste trabalho serão apresentados com mais detalhes. O primeiro descreve o sistema Hybrid que é um SMA para gerência de redes de telecomunicações. O segundo descreve o sistema Victor que é um SMA para gerência de falhas desenvolvido para uma rede internacional de telecomunicações. Estes dois trabalhos são voltados à gerência de redes de telecomunicações (WANs). Por fim, o terceiro trabalho descreve um arcabouço (*framework*) sem nome desenvolvido por Lavinal *et al.* para a criação de redes auto-gerenciáveis através do uso de um SMA. Este trabalho é mais aplicado para a gerência de redes de computadores locais (LANs).

5.2 Sistema Hybrid

O sistema Hybrid [Evans99] é um sistema de gerência de redes ATM [I.321] desenvolvido por pesquisadores da empresa Broadcom Eireann Research. O principal objetivo do sistema é melhorar a escalabilidade da gerência de redes ATM. Para tanto é utilizado um método de distribuição de tarefas de gerência entre um conjunto de controladores espalhados pela rede e que são baseados em agentes de software. Cada controlador é responsável por um conjunto de recursos da rede e interage com controladores de nível mais baixo para prover serviços a clientes ou controladores de nível superior. A Figura 5.1 ilustra uma hierarquia em três níveis dos tais controladores que também são conhecidos como *autoridades*. Cada autoridade executa as tarefas típicas de gerência (FCAPS) e também se comporta como um agente de software trocando mensagens KQML [Finin97] com outras autoridades.

Cada autoridade é por si só um sistema multi-agentes. A Figura 5.2 ilustra o modelo de uma autoridade, composta de oito tipos de agentes. Há um tipo de agente responsável por cada uma das áreas funcionais de gerência (FCAPS). Além destes cinco agentes, há um agente de Serviço responsável pela interação com agentes de clientes e com agentes de outras autoridades. Há ainda um agente de Recurso que fornece uma interface entre os recursos da rede e os demais agentes do sistema e um agente ACA (*Authority Controlling Agent*) que controla a plataforma como um todo.

Cada agente de software do sistema é implementado na linguagem de sistemas especialistas CLIPS [Giarratano04]. Há dois módulos principais em cada agente; um é responsável pelas comunicações do mesmo com outros agentes e o outro controla as interações com recursos de rede sob a responsabilidade do agente

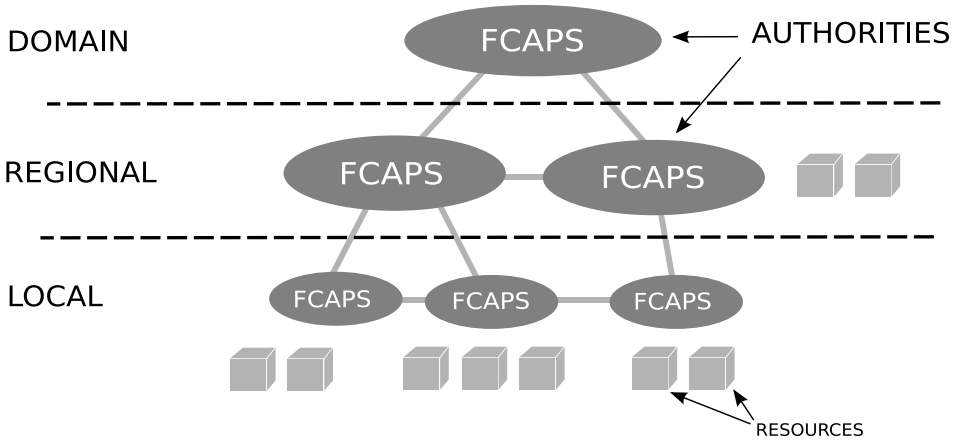

Figura 5.1: Arquitetura de autoridades

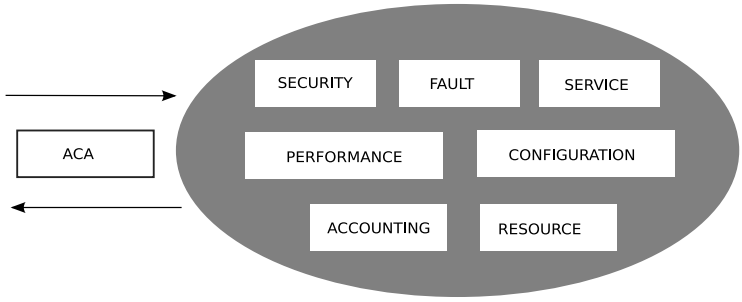

Figura 5.2: Visão conceitual de uma autoridade

em questão. A base de conhecimento de cada agente é dividida em três modelos relacionados a diferentes domínios, a saber, (i) o modelo próprio (*self model*), que armazena as metas que o agente possui, (ii) o modelo do mundo (*world model*), que mantém informações sobre objetos conhecidos pelo agente, e (iii) o modelo de agentes (*agent model*), que descreve o que o agente conhece a respeito de outros agentes. Além destes modelos, que mantêm as crenças dos agentes, há ainda uma base de habilidades (*skills*) que são os *scripts* utilizados pelos agentes para atingirem suas metas. Cada *script* de habilidade tem um conjunto de pré-condições que devem ser satisfeitas antes de ser executado, assim como um custo associado. Com isto, o agente pode escolher a ação mais adequada entre o conjunto de ações disponíveis.

Um dos principais recursos gerenciados pelo sistema Hybrid são circuitos utilizados por clientes. Cada circuito está diretamente associado a uma autoridade que se torna responsável pelo circuito no sistema. A autoridade associada a um circuito é a de nível mais baixo que contém ambas as terminações do circuito. A interação entre o cliente (usuário do circuito) e a autoridade associada ao circuito é desempenhada através de um agente *Proxy* que representa os interesses do cliente na autoridade em questão.

Quando, por exemplo, um cliente solicita a uma autoridade o estabelecimento de um circuito, ou uma mudança de configuração em um circuito já estabelecido, a autoridade verifica primeiramente quais outros recursos (equipamentos) e sub-autoridades serão afetados. Nos casos em que o roteamento do circuito puder ser feito através de diferentes sub-autoridades, o sistema deve decidir quais são as mais apropriadas. Para possibilitar esta escolha, cada autoridade exporta índices de desempenho que indicam a "saúde" da sub-rede levando-se em consideração aspectos como congestionamento, retardo e número de falhas, por exemplo. Esta interação entre autoridades e sub-autoridades pode ser encarada como uma espécie de negociação entre as mesmas. Este mecanismo permite uma distribuição mais uniforme dos serviços pelas sub-autoridade sem que nenhuma seja sobrecarregada, o que contribui para uma melhor qualidade dos serviços prestados pela rede.

Outra dinâmica interessante do sistema Hybrid ocorre quando um problema ocorre. Nestes casos, o agente de desempenho (que detectou a falha) informa o agente de configuração sobre o problema. Este pode, então, optar por uma de três alternativas: (i) resolver o problema (caso seja possível), (ii) modificar os serviços afetados de forma acordada com os respectivos agentes de clientes, ou (iii) sinalizar a falha para a autoridade de nível superior. A vantagem deste mecanismo é que em muitos casos os problemas são resolvidos em nível local sem a necessidade de envolvimento de autoridades superiores.

5.3 Sistema Victor

O sistema Victor [Odubiyi01] é um sistema experimental para gerência de falhas em uma rede baseada nas tecnologias ATM [I.321] e Frame Relay [I.233] então mantida pela empresa Concert Communications Services. O sistema Victor tem seis objetivos principais, listados a seguir:

- Exibir a topologia da rede e indicar problemas em elementos da rede através de uma interface gráfica;
- Detectar a localização de falhas em circuitos (VCs) específicos para os quais foram levantados *trouble tickets*;
- Disponibilizar relatórios de nível gerencial sobre falhas em circuitos e em recursos associados (como nós, cartões e enlaces, por exemplo);
- Usar mecanismos de predição de falhas para antecipar eventuais degradações no desempenho de equipamentos;
- Adaptar algoritmos de tendência (*trending algorithms*) existentes para avaliar estatísticas de desempenho de equipamentos de rede e dar suporte a uma gerência pró-ativa de falhas;

– Dar suporte ao cumprimento de metas de desempenho de rede (QoS) através do monitoramento dos elementos da rede.

O sistema Victor é fortemente baseado em agentes de software cooperativos que interagem para alcançar os objetivos levantados acima. Os agentes são divididos em cinco camadas diferentes, a saber, (i) a de interface de usuário, que contém os agentes responsáveis pela interação com os operadores de rede, (ii) a de gerência de serviço de agentes, que tem um agente que oferece serviços de infra-estrutura (tais como "páginas amarelas") aos demais agentes, (iii) a de resolução de problemas, que contém três tipos de agentes especializados na identificação, na previsão e no registro de falhas, (iv) a de monitoramento de rede e de desempenho, cujos agentes são responsáveis pelo monitoramento de recursos de rede e pela coleta de dados usados pelos agentes da camada de resolução de problemas, e (v) a de interface com fontes de dados de gerência, cujos agentes servem como intermediários na comunicação entre o sistema Victor e os sistemas de gerência legados. No sistema Victor, as informações de estatísticas de operação dos recursos de rede são obtidas diretamente dos mesmos via CLI, e também via um *data warehouse* que é periodicamente atualizado. Os dados de topologia, por sua vez, são provenientes de uma base relacional centralizada.

Os agentes do sistema Victor comunicam-se através da linguagem FIPA ACL [FIPA00061]. Alguns dos agentes foram desenvolvidos com a plataforma Zeus [Nwana99b]. Também foi utilizada a linguagem JESS [FriedmanHill03] para a implementação de agentes.

5.4 Arcabouço de Lavinal *et al.*

O arcabouço de Lavinal *et al.* [Lavinal06] tem por objetivo a construção de redes auto-gerenciáveis através de uso de agentes de software. O arcabouço conceitual proposto consiste de grupos de agentes de gerência que possuem habilidades específicas e que estão associados a recursos gerenciáveis da rede. Tais grupos compartilham uma representação comum do ambiente de rede a eles associado. Além disto, as funcionalidades de gerência são modeladas através de uma ontologia própria para este domínio. Através da análise deste modelo da rede e desta ontologia, comportamentos locais (internos aos agentes) e sociais (que envolvem interações entre agentes) podem ser deduzidos e disparados, o que proporciona autonomia da rede no que se refere à sua gerência.

O arcabouço apresenta alguns conceitos organizacionais de SMA adaptados ao domínio de gerência de redes. Tais conceitos são fundamentais para a proposta e são apresentados a seguir:

Elemento gerenciado (EG). Corresponde a um recurso real que deve ser gerenciado. Tal recurso pode ser hardware ou software;

Ambiente gerenciado. Corresponde a um conjunto de EGs relacionados entre si. Um possível exemplo é uma rede local de computadores;

Agente de gerência (AG). Especialização de um agente de software para o domínio de gerência. Possui um conjunto de habilidades que o permitem perceber e atuar sobre EGs;

Acoplamento AG-EG. Indica o relacionamento entre um AG e um EG. Quando há o acoplamento, o AG se torna responsável por algum aspecto da gerência do EG;

Funcionalidade genérica de gerência (FGG). Funcionalidades típicas de gerência tais como monitoramento, configuração, correção, otimização e proteção. Estas funcionalidades são desempenhadas por AGs;

Papel de um AG. Indica tanto a funcionalidade de gerência um AG está desempenhando assim como o EG em que o AG está atuando;

Grupo de AGs. Indica o grupo de AGs responsáveis por EG de um ambiente gerenciado;

Sistema auto-gerenciável. Representa um conjunto com um ou mais grupos de AGs e os respectivos ambientes gerenciados.

Segundo este arcabouço cada AG é responsável por um aspecto da gerência de um ou mais EGs. Estes aspectos nada mais são do que funcionalidades existentes no conjunto de FGGs disponíveis.

Para permitir que os AGs deliberem sobre o ambiente e interajam uns com os outros, este trabalho propõe tanto uma estratégia para modelar ambientes gerenciáveis como o uso de ontologias para modelar as funcionalidades de gerência.

Para modelar o ambiente, é proposto o uso de três categorias distintas de EGs, a saber, (i) *serviço*, que representa serviços de software, (ii) *sistema final*, que representa os equipamentos que hospedam os serviços, e (iii) *sistema intermediário*, que representa elementos que manipulam fluxos de dados. A categoria de sistemas intermediários é subdividida em três outras subcategorias de acordo com o tipo de manipulação de fluxo de dados que se desempenha. Estas três categorias são (i) *canal*, que apenas propaga fluxos de dados às entidades vizinhas (cabos ou enlaces de rede, por exemplo), (ii) *filtro*, que bloqueia ou redireciona fluxos de dados (*firewall*, por exemplo), e (iii) *transformador*, que modifica um fluxo de dados (serviço NAT, por exemplo).

Além de modelar os EGs em si, também é necessário modelar os relacionamentos entre os mesmos. No trabalho são identificados três tipos de relacionamentos também denominados dependências: (i) dependência de serviço, usada para modelar a dependência entre dois serviços de software (um servidor de aplicações que depende de um servidor de DNS, por exemplo), (ii) dependência de hospedagem, usada para modelar o fato de que um serviço está hospedado em um sistema final, e (iii) dependência de fluxo, que representa um enlace entre sistemas finais e sistemas intermediários.

Para modelar as funcionalidades de gerência desempenhadas pelos AGs, este trabalho propõe o uso de ontologias projetadas especificamente para o domínio de gerência de redes. Tais ontologias permitem a um AG declarar as ações de gerência que ele é capaz de desempenhar e incluir em seus processos deliberativos as ações que devem ser executadas por outros agentes para que seus objetivos próprios sejam atingidos.

Para que estas ontologias de gerência sejam compartilhadas por diferentes AGs em um grupo, o trabalho propõe o uso de um agente de ontologia cuja principal função é centralizar e manter as ontologias. Os demais agentes do sistema podem consultar o agente de ontologia para identificar os papéis que implementam uma ação de gerência específica. Também é proposto o uso de um agente de grupo que centraliza as informações sobre os papéis desempenhados por todos os AGs em um grupo de AGs. Este agente de grupo é consultado quando um AG deseja encontrar outro AG para lhe solicitar alguma ação ou informação de gerência.

5.5 Avaliação dos Sistemas Apresentados

5.5.1 Sistema Hybrid

O sistema Hybrid é bastante interessante devido ao amplo escopo que ele abrange no contexto de gerência de redes. Como foi mencionado na apresentação acima, o sistema contempla todas as cinco grandes áreas de gerência de redes (FCAPS), contando com agentes específicos para cada uma das áreas. Teoricamente, portanto, seria possível administrar por completo uma rede de telecomunicações com este sistema.

Uma interessante contribuição deste sistema é o fato de que cada agente de software do mesmo mantém bases de conhecimento (chamadas de modelos) sobre si próprio, sobre a rede e sobre os demais agentes. Tais bases são fundamentais para garantir os aspectos de autonomia e pró-atividade dos agentes de software deste

sistema. Tal autonomia é evidenciada na dinâmica para resolução automática de falhas discutidas na apresentação do sistema.

Outro importante aspecto que deve ser mencionado é a existência de uma base de habilidades, que nada mais são do que os mecanismos que os agentes possuem para se comunicar com os equipamentos de rede, tanto para deles extrair informações quanto para agir sobre os mesmos. Esta abstração é equivalente às abstrações de Testes e Tarefas da arquitetura proposta neste trabalho discutidas nos capítulos anteriores.

Outra característica interessante do sistema Hybrid, e que é indicada pelo seu próprio nome, é que este adota uma arquitetura híbrida (ou hierárquica) para a gerência de redes. Em outras palavras, ele adota um modelo intermediário entre as alternativas centralizada e distribuída. Tal característica é semelhante ao modelo da arquitetura proposta.

Todavia, uma desvantagem observada neste sistema é decorrente justamente do amplo escopo que o sistema pretende abranger. Como trata-se de uma solução completa para a gerência de redes, integrá-lo com sistemas legados de gerência utilizados na prática em redes em produção pode ser uma tarefa bastante complexa. Há naturalmente uma grande área de interseção e conflitos com os sistemas pré-existentes, o que pode dificultar sua coexistência. Isto é evidenciado pela dinâmica de gerência do estabelecimento de circuitos descrita em sua apresentação. Tal responsabilidade é tipicamente delegada diretamente aos sistemas legados de gerência. Como foi visto, a arquitetura proposta neste trabalho foi concebida com o intuito de adequar-se a redes legadas pré-existentes, minimizando o impacto na operação e funcionamento da rede.

5.5.2 Sistema Victor

O sistema Victor tem uma proposta menos ousada e abrangente que o sistema Hybrid. Ao invés de englobar todas as áreas de gerência de redes (FCAPS), os idealizadores do sistema Victor focaram seus esforços apenas nas áreas de gerência de falhas e performance. Este foco fica claro a partir dos seis principais objetivos do sistema relacionados quando de sua apresentação. O fato de que havia uma rede em produção que devia ser mantida em operação também sugere que a pesquisa teve que manter um compromisso com soluções mais pragmáticas e que causassem o menor impacto possível na operação da rede.

Tal qual o sistema Hybrid e a solução proposta neste trabalho, este sistema também segue um paradigma hierárquico para a gerência de redes. Há componentes centralizados, como bases de dados que mantêm dados sobre a topologia da rede assim como bases com o histórico de falhas de elementos da rede. Também há

componentes distribuídos, notoriamente os agentes de software que desempenham atividades de monitoramento de equipamentos e correção de falhas.

Este trabalho é um exemplo inequívoco de que tecnologias baseadas em agentes de software podem ser realmente utilizados em situações práticas e que elas não estão restritas apenas a redes experimentais em laboratórios de instituições de pesquisa.

Não há menção, entretanto, sobre estratégias para distribuição de informações pela rede. A ausência deste tipo de funcionalidade pode limitar o tipo de testes assim como a freqüência com que são efetuados sobre a rede. Como foi visto anteriormente, esta é uma questão central da arquitetura proposta. Outra questão é que os tipos de teste que este sistema parece suportar são relativamente simples, baseados apenas em monitoramento de estatísticas. Como foi discutido em capítulos anteriores, a arquitetura proposta é bem mais flexível no que diz respeito ao tipo de testes com os quais é capaz de lidar.

5.5.3 Arcabouço de Lavinal *et al.*

O trabalho de Lavinal *et al.* é o mais recente dos três apresentados neste capítulo, e é o que propõe um embasamento teórico mais sólido no que diz respeito ao uso de agentes de software no domínio de gerência de redes. Sua principal contribuição é a modelagem de uma base de conhecimento (ontologia) para a gerência de redes. Algo semelhante também é mencionado no texto sobre o sistema Hybrid, a saber, os modelos (próprio, de mundo e de agentes) e a base de habilidades, mas este tema é ali abordado de forma bastante superficial. A discussão do trabalho de Lavinal *et al.* é, entretanto, bastante detalhada.

Uma contribuição interessante é a adaptação dos conceitos de agentes de rede ao domínio de gerência de redes. Este arcabouço teórico define as principais entidades e relacionamentos que devem ser considerados em implementações de sistemas multi-agentes aplicados à gerência de redes. A ontologia apresentada contempla a modelagem de recursos de rede, como equipamentos, enlaces e serviços de software, assim como as relações de topologia e dependência entre serviços e equipamentos. São, ainda, definidas ontologias para a modelagem de ações de gerência. Tais abstrações permitem a construção de agentes deliberativos, conferindo mais pró-atividade e autonomia a tais sistemas de gerência.

Este arcabouço possui um escopo amplo, no sentido de que contempla todas as cinco grandes áreas de gerência de redes (FCAPS). Ele é também bastante genérico e, por isso, parece ser mais flexível do que o sistema Hybrid para uso em redes reais. Esta flexibilidade seria bastante útil para evitar conflitos com sistemas de gerência pré-existentes.

O paradigma de gerência adotado neste trabalho é o distribuído. Isto é compreensível uma vez que o seu foco é em redes de computadores locais (baseadas no protocolo IP), que são por essência mais distribuídas que redes baseadas em tecnologias legadas de telecomunicações. Não é comum, por exemplo, encontrar em redes locais um elemento central de gerência da rede. Portanto, não há neste trabalho menção a algum tipo de política para lidar com elementos centralizados como os comumente encontrados nas redes de telecomunicações discutidas no presente trabalho. Por conta disso, tal solução carece de adaptações para ser aplicada a este tipo de redes.

6
Considerações Finais e Trabalhos Futuros

A sociedade moderna tem se tornado cada vez mais dependente de serviços de telecomunicações como telefonia e acesso à Internet. Tais serviços são disponibilizados através de redes de telecomunicações cada vez maiores e mais complexas, e que são compostas por equipamentos de diferentes tipos e tecnologias. Para administrar tais redes de forma eficiente e economicamente competitiva, diferentes modelos e estratégias de gerência de redes têm sido desenvolvidos e empregados ao longo do tempo. Cada novo modelo é construído com base nos sucessos e fracassos dos modelos anteriores, num campo de pesquisa que tem se mostrado bastante dinâmico.

Este trabalho explorou alguns dos principais conceitos da área de gerência de redes, apresentando os mais importantes e tradicionais padrões de gerência, assim como novas tecnologias que têm sucedido estes padrões e que são baseadas em abstrações tais como objetos distribuídos e código móvel, por exemplo. Uma ênfase especial foi dada ao uso de agentes de software aplicados ao domínio de gerência de redes. A utilização de agentes de software é bastante apropriada neste contexto em que pró-atividade, autonomia e flexibilidade são buscados com tamanha propriedade.

Neste trabalho foi apresentada uma arquitetura baseada em agentes de software para a gerência de falhas em redes legadas de telecomunicações. Foram também exibidos dois estudos de caso que ilustram como tal arquitetura pode ser utilizada para a modelagem de testes. O objetivo principal da arquitetura é o de possibilitar o monitoramento e a aplicação de rotinas para o diagnóstico e a correção de falhas em redes gerenciadas por sistemas centralizados. Destaque deve ser dado para o fato de que tais testes pode ser feitos de forma distribuída. Pode-se considerar que este objetivo faz parte de uma meta de escopo mais abrangente que é a de possibilitar que tais redes legadas satisfaçam novos e mais exigentes requisitos de operação. Como foi discutido no decorrer do trabalho, esta meta é relevante para muitas organizações pois tais redes podem continuar em operação por muitos anos mesmo depois de serem consideradas antiquadas e ultrapassadas por soluções baseadas em tecnologias mais modernas. Não é uma tarefa árdua imaginar-se um exemplo de como tal arquitetura pode ser útil. Um sistema com

testes eficazes para a detecção de falhas construído sobre a arquitetura proposta pode diagnosticar falhas rapidamente, ou mesmo previni-las, contribuindo para um menor tempo de indisponibilidade da rede e, conseqüentemente, alcançando um melhor nível de qualidade do serviço exigido por clientes ou imposto pelo mercado. Caso a permanência em operação de tais redes legadas seja de fato necessária, soluções como a aqui apresentada podem contribuir no sentido de aumentar sua vida útil. Isto se torna possível na medida em que novos requisitos operacionais podem ser satisfeitos, e a defasagem em relação a redes mais modernas pode ser diminuída.

Naturalmente, para que tais objetivos sejam alcançados com êxito, algumas restrições devem ser observadas. Uma das mais importantes é que sistemas baseados na arquitetura proposta devem exercer o mínimo de carga sobre o sistema de gerência existente, de modo a não prejudicar o seu funcionamento regular. Para tanto, foi proposto um mecanismo que distribui informações de topologia e inventário por agentes de software distribuídos pela rede. Isto reduz a freqüência necessária de interação com o sistema de gerência e, conseqüentemente, proporciona um menor impacto sobre o mesmo. Para tanto, a rede é dividida em partições, definidas por critérios geográficos. Para cada partição existe um agente responsável por fornecer dados de topologia para os demais agentes do sistema.

Outra importante contribuição deste trabalho é a modelagem de ações de diagnóstico de falhas através de testes e tarefas. O uso destas abstrações, em particular a das tarefas, permite uma maior reutilização de código em sistemas de diagnósticos de falhas em redes de telecomunicações. Isto é possível uma vez que as comunicações com equipamentos de rede são isoladas em elementos atômicos com interfaces de entrada e saída bem definidas. Assim sendo, cada tarefa pode ser facilmente utilizada em várias situações, o que pode diminuir os esforços de criação de testes novos. Além do benefício da reutilização de código, outra importante característica desta modelagem baseada em testes e tarefas é a sua flexibilidade no que diz respeito ao tipo de ações que podem ser implementados através dela. Como há poucas restrições em relação à implementação das tarefas, praticamente qualquer tipo de comunicação com equipamentos pode ser desempenhado. Não há restrição, por exemplo, ao tipo de protocolo de gerência utilizado para tal comunicação. Pode-se utilizar protocolos como SNMP e TL1, ou mesmo CLI. Quando da implementação de um teste, além da flexibilidade obtida através das tarefas, há ainda a possibilidade da obtenção de informações de topologia, o que expande ainda mais o universo de testes que este modelo é capaz de contemplar.

Por fim, uma última questão é que, apesar de este sistema ter sido concebido com o foco na área de gerência de falhas, também é possível utilizá-lo sem praticamente nenhuma alteração em outras áreas de gerência de redes, como a

gerência de configuração, por exemplo. Tal característica se deve à flexibilidade obtida através dos testes e tarefas, que podem ser adaptados para ações de gerência e não apenas diagnóstico e correção de falhas.

Apesar de a arquitetura aqui proposta atender aos objetivos inicialmente estabelecidos, há ainda bastante espaço para melhorias da mesma. Algumas destas melhorias são identificadas e discutidas a seguir.

A questão possivelmente mais importante que não foi considerada no escopo deste trabalho foi a criação e a aplicação de uma ontologia para modelagem dos recursos de rede e dos testes que podem ser efetuados sobre a mesma. Esta ontologia seria uma peça fundamental para possibilitar interações mais ricas do ponto de vista semântico entre os agentes de software. Parte deste problema foi solucionada neste trabalho com o uso de esquemas XML para modelagem dos recursos de rede. Pouco foi feito, entretanto, para modelar os testes de forma que os agentes pudessem ponderar ou deliberar sobre os mesmos. O uso de ontologias seria uma condição necessária para que os agentes de software, que neste trabalho são fundamentalmente reativos, pudessem assumir também características deliberativas. Um bom ponto de partida para esta questão de ontologias de gerência de redes pode ser encontrado em [Lavinal06].

Outra área em que há espaço para avanços é a criação de mecanismos para que o local de execução de um teste possa migrar. Isto seria útil em testes em que a comunicação com equipamentos localizados em partições diferentes da rede é necessária. No estado atual, limitações da arquitetura exigem que toda execução de um teste seja conduzida a partir de um ponto único. Este ponto é necessariamente um agente de teste, que é imóvel. Uma alternativa para a resolução deste problema seria permitir que os agentes de testes fossem móveis, deslocando-se pela rede. Outra alternativa seria criar um mecanismo para a transferência da execução de testes entre agentes de testes.

Outro aspecto em que há espaço para melhorias diz respeito ao fato de que os agentes de testes modelados neste trabalho estão preparados para coletar informações de equipamentos e sistemas de gerência, mas não para receber alarmes e notificações dos mesmos. Há espaço aqui para a criação de um novo tipo de agente de software que possua mecanismos para monitorar alarmes de equipamentos, *logs* de sistemas de gerência e similares e que tome as devidas ações em função destas informações coletadas. Esta estratégia ajudaria a introduzir mais autonomia na arquitetura proposta.

Por fim, uma última frente para novos desenvolvimentos seria a pesquisa e aplicação de algoritmos e estratégias para previsão de falhas e a adaptação dos mesmos no contexto da arquitetura proposta neste trabalho.

Referências Bibliográficas

[Albayrak98a] ALBAYRAK, Sahin. **Intelligent Agents for Telecommunications Applications**. IOS Press, 1998. 5.1

[Albayrak98b] ALBAYRAK, Sahin; GARIJO, Francisco. **Proceedings of the Second International Workshop on Intelligent Agents for Telecommunications Applications**, Paris, França – IATA'98. Springer, 1998. 5.1

[Albayrak99] ALBAYRAK, Sahin. **Proceedings of the Third International Workshop on Intelligent Agents for Telecommunications Applications**, Stockholm, Suécia – IATA'99. Springer, 1999. 5.1

[Alur03] ALUR, Deepak; MALKS, Dan; CRUPI, John. **Core J2EE Patterns: Best Practices and Design Strategies**. Segunda edição, Prentice Hall, 2003. 3.4.2

[Barr93] BARR, William; BOYD, Trevor; INOUE, Yuji. The TINA Initiative. **IEEE Communications Magazine**, v. 31, n. 3, pp. 70-76, 1993. 2.1.2

[Bieszczad98] BIESZCZAD, Andrzej; PAGUREK, Bernard; WHITE, Tony. Mobile Agents for Network Management. **IEEE Communications Surveys & Tutorials**, v. 1, n. 1, pp. 2-9, 1998. 5.1

[Boutaba04] BOUTABA, Raouf; GOLAB, Wojciech; IRAQI, Youssef. Lightpaths on Demand: A Web Services-Based Management System. **IEEE Communications Magazine**, v. 42, n. 7, pp. 101-107, 2004. 2.1.4

[Bratman87] BRATMAN, Michael. **Intentions, Plans and Practical Reasons**. Harvard University Press, 1987. 2.2.4

[Bregni02] BREGNI, Stefano. **Synchronization of Digital Telecommunications Networks**. John Wiley & Sons, 2002. 4.2

[Cheikhrouhou98] CHEIKHROUHOU, Morsy; CONTI, Pierre; LABETOULLE, Jacques. Intelligent Agents in Network Management, A State of the Art. **Networking and Information Systems Journal**, v. 1, n. 1, pp. 9-38, 1998. 5.1

[Chen02] CHEN, Thomas; LIU, Stephen. A Model and Evaluation of Distributed Network Management Approaches. **IEEE Journal on Selected Areas in Communications**, v. 20, n. 4, pp. 850-857, 2002. 2.1.3

[Choi04] CHOI, Mi-Jung; CHOI, Hyoun-Mi; HONG, James. XML-Based Configuration Management for IP Network Devices. **IEEE Communications Magazine**, v. 42, n. 7, pp. 84-91, 2004. 2.1.4

[CIM] Common Information Model. Disponível em: http://www.dmtf.org/standards/cim. 2.1.2

[Cisco03] Cisco Systems, Inc. **Internetworking Technologies Handbook**. Quarta edição, Cisco Press, 2003. 2.1.2, 2.1.2

[Dale01] DALE, Jonathan; MAMDANI, Ebrahim. Open Standards for Interoperating Agent-Based Systems. **Software Focus**, v. 2, n. 1, pp. 1-8, 2001. 2.2.5

[DeMeer00] DE MEER, Hermann; LA CORTE, Aurelio; PULIAFITO, Antonio. Programmable Agents for Flexible QoS Management in IP Networks. **IEEE Journal on Selected Areas in Communications**, v. 18, n. 2, pp. 256-267, 2000. 5.1

[DTD] Document Type Definition, W3C. Disponível em: http://www.w3.org/TR/REC-xml/#dt-doctype. 2.1.4

[ElSayed02] EL-SAYED, Mohamed; JAFFE, Jeffrey. A View of Telecommunications Network Evolution. **IEEE Communications Magazine**, v. 40, n. 12, pp. 74-81, 2002. 2.1.1

[Emmerich02] EMMERICH, Wolfgang. Distributed Component Technologies and their Software Engineering Implications. Em: **Proceedings of the 24th International Conference on Software Engineering**, Orlando, EUA – ICSE 2002. ACM Press, pp. 537-546, 2002. 2.2.1

[Evans99] EVANS, Richard; SOMERS, Fergal; KERR, David; O'SULLIVAN, Donie. A Multi-Agent System Architecture for Scalable Management of High Performance Networks: Applying Arms Length Autonomy. **Software Agents for Future Communications Systems**, Springer, cap. 3, pp. 86-111, 1999. 5.2

[Finin97] FININ, Tim; LABROU, Yannis; MAYFIELD, James. KQML as an Agent Communication Language. **Software Agents**, AAAI Press, pp. 291-316, 1997. 2.2.5, 5.2

[FIPA00001] FIPA00001. **FIPA Abstract Architecture Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00023] FIPA00023. **FIPA Agent Management Specification**. Foundation for Intelligent Physical Agents, 2004. 2.2.5, 3.4.3

[FIPA00026] FIPA00026. **FIPA Request Interaction Protocol Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5, 3.5

[FIPA00027] FIPA00027. **FIPA Query Interaction Protocol Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5, 3.5

[FIPA00037] FIPA00037. **FIPA Communicative Act Library Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00061] FIPA00061. **FIPA ACL Message Structure Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5, 3.5, 5.3

[FIPA00067] FIPA00067. **FIPA Agent Message Transport Service Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00069] FIPA00069. **FIPA ACL Message Representation in Bit-Efficient Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00070] FIPA00070. **FIPA ACL Message Representation in String Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00071] FIPA00071. **FIPA ACL Message Representation in XML Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00075] FIPA00075. **FIPA Agent Message Transport Protocol for IIOP Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00076] FIPA00076. **FIPA Agent Message Transport Protocol for WAP Specification**. Foundation for Intelligent Physical Agents, 2001. 2.2.5

[FIPA00082] FIPA00082. **FIPA Network Management and Provisioning Specification**. Foundation for Intelligent Physical Agents, 2001. 2.2.5, 5.1

[FIPA00084] FIPA00084. **FIPA Agent Message Transport Protocol for HTTP Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00085] FIPA00085. **FIPA Agent Message Transport Envelope Representation in XML Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[FIPA00088] FIPA00088. **FIPA Agent Message Transport Envelope Representation in Bit Efficient Specification**. Foundation for Intelligent Physical Agents, 2002. 2.2.5

[Fowler04] FOWLER, Martin. **Inversion of Control Containers and the Dependency Injection Pattern**. 2004. Disponível em: http://martinfowler.com/articles/injection.html. 3.5

[Franklin96] FRANKLIN, Stan; GRAESSER, Art. Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. Em: **Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages**. Springer, 1996. 2.2.2

[FriedmanHill03] FRIEDMAN-HILL, Ernest. **Jess in Action: Rule-Based Systems in Java**. Manning Publications, 2003. 5.3

[Fuggetta98] FUGGETTA, Alfonso; PICCO, Gian; VIGNA, Giovanni. Understanding Code Mobility. **IEEE Transactions on Software Engineering**, v. 24, n. 5, pp. 342-361, 1998. 2.1.3

[G.707] ITU-T Recommendation G.707. **Network Node Interface for the Synchronous Digital Hierarchy (SDH)**, 2003. 2.1.2

[Gamma95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley, 1995. 3.4.2

[Giarratano04] GIARRATANO, Joseph; RILEY, Gary. **Expert Systems: Principles and Programming**. Course Technology, 2004. 5.2

[Gibney01] GIBNEY, M.; JENNINGS, N.; VRIEND, N.; GRIFFITHS, J. Market-Based Call Routing in Telecommunications Networks Using Adaptive Pricing and Real Bidding. **Agent Technology for Communication Infrastructures**, John Wiley & Sons, cap. 18, pp. 234-248, 2001. 5.1

[Glitho95] GLITHO, Roch; HAYES, Stephen. Telecommunications Management Network: Vision vs. Reality. **IEEE Communications Magazine**, v. 33, n. 3, pp. 47-52, 1995. 2.1.2

[Goldszmidt95] GOLDSZMIDT, Germán; YEMINI, Yechiam. Distributed Management by Delegation. Em: **Proceedings of the 15th International Conference on Distributed Computing Systems**, Vancouver, Canadá – ICDCS 1995. IEEE Computer Society Press, pp. 333-340, 1995. 2.1.3

[González02] GONZÁLEZ-VALENZUELA, Sergio; LEUNG, Victor. QoS Routing for MPLS Networks Employing Mobile Agents. **IEEE Network**, v. 16, n. 3, pp. 16-21, 2002. 5.1

[Gosling05] GOSLING, James; JOY, Bill; STEELE, Guy; BRACHA, Gilad. **The Java Language Specification**. Terceira edição, Prentice Hall, 2005. 3.5

[GPS] Global Positioning System. Disponível em: http://www.gps.gov. 4.2

[GR-831] GR-831-CORE. Telcordia Technologies, Inc. **Operation Application Messages – Language for Operations Application Messages**, 1996. 2.1.1, 3.3.2

[Hansen01] HANSEN, M.; JENSEN, P.; SOLDATOS, J.; VAYIAS, E. Low-Level Control of Network Elements from an Agent Platform. **Agent Technology for Communication Infrastructures**, John Wiley & Sons, cap. 12, pp. 156-166, 2001. 5.1

[Hayzelden99a] HAYZELDEN, Alex; BIGHAM, John. **Software Agents for Future Communications Systems**, Springer, 1999. 5.1

[Hayzelden99b] HAYZELDEN, Alex; BIGHAM, John; WOOLDRIDGE, Michael; CUTHBERT, Laurie. Future Communication Networks Using Software Agents. **Software Agents for Future Communications Systems**, Springer, cap. 1, pp. 1-57, 1999. 2.2.3, 2.2.4

[Hayzelden00] HAYZELDEN, Alex; BIGHAM, John; POSLAD, Stefan; BUCKLE, Phil; MAMDANI, Ebrahim. Communications Systems Driven by Software Agent Technology. **Journal of Network and Systems Management**, v. 8, n. 3, pp. 321-345, 2000. 5.1

[Hayzelden01] HAYZELDEN, Alex; BOURNE, Rachel. **Agent Technology for Communication Infrastructures**. John Wiley & Sons, 2001. 5.1

[Horlait00] HORLAIT, Eric. **Proceedings of the Second International Workshop on Mobile Agents for Telecommunications Applications**, Paris, França – MATA 2000. Springer, 2000. 5.1

[Horlait03] HORLAIT, Eric; MAGEDANZ, Thomas; GLITHO, Roch. **Proceedings of the Fifth International Workshop on Mobile Agents for Telecommunications Applications**, Marakech, Marrocos – MATA 2003. Springer, 2003. 5.1

[I.233] ITU-T Recommendation I.233. **Frame Mode Bearer Services**, 1991. 3.2, 5.3

[I.321] ITU-T Recommendation I.321. **B-ISDN Protocol Reference Model and its Application**, 1991. 2.1.1, 3.2, 5.1, 5.2, 5.3

[ISO7498] ISO/IEC 7498-1:1994. **Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model**, 1994. 2.1.1, 2.1.2

[JADE] Java Agent Development Framework. Disponível em: http://jade.tilab.com. 1, 3.5

[JAXB] Java Architecture for XML Binding. Disponível em: http://java.sun.com/webservices/jaxb. 3.4.2, 3.5

[JaxMe] JaxMe 2. Disponível em: http://ws.apache.org/jaxme. 3.5

[Jennings01] JENNINGS, Nicholas. An Agent-Based Approach for Building Complex Software Systems. **Communications of the ACM**, v. 44, n. 4, pp. 35-41, 2001. 2.2.3

[JIDM] Joint Inter-Domain Management. Disponível em: http://www.opengroup.org/external/jidm. 2.1.3

[JIDM00] Joint Inter-Domain Management. **Inter-Domain Management: Specification and Interaction Translation**. The Open Group, 2000. 2.1.3

[Juniper] Juniper Networks. **JUNOScript API Software**. Disponível em: http://juniper.net/support/xml/junoscript. 2.1.4, 3.3.2

[Karmouch99] KARMOUCH, Ahmed; IMPEY, Roger. **Proceedings of the First International Workshop on Mobile Agents for Telecommunications Applications**, Ottawa, Canadá – MATA 1999. World Scientific Pub, 1999. 5.1

[Karmouch02] KARMOUCH, Ahmed; MAGEDANZ, Thomas; DELGADO, Jaime. **Proceedings of the Forth International Workshop on Mobile Agents for Telecommunications Applications**, Barcelona, Espanha – MATA 2002. Springer, 2002. 5.1

[Kohli03] KOHLI, Madhur; LOBO, Jorge. Realizing Network Control Policies Using Distributed Action Plans. **Journal of Network and Systems Management**, v. 11, n. 3, pp. 305-327, 2003. 2.1.4

[Labrou99] LABROU, Yannis; FININ, Tim; PENG, Yun. Agent Communication Languages: The Current Landscape. **IEEE Intelligent Systems**, v. 14, n. 2, pp. 45-52, 1999. 2.2.5

[Lavinal06] LAVINAL, Emmanuel; DESPRATS, Thierry; RAYNAUD, Yves. A Generic Multi-Agent Conceptual Framework Towards Self-Management. Em: **Proceedings of the 10th Network**

Operations and Management Symposium, Vancouver, Canadá – NOMS 2006. IEEE, pp. 394-403, 2006. 5.4, 6

[Lazar97] LAZAR, Aurel. Programming Telecommunication Networks. **IEEE Network**, v. 11, n. 5, pp. 8-18, 1997. 2.1.3

[Liotta99] LIOTTA, Antonio; KNIGHT, Graham; PAVLOU, George. On the Performance and Scalability of Decentralised Monitoring Using Mobile Agents. Em: **Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management**, Zurique, Suíça – DSOM'99. Springer, pp. 3-18, 1999. 2.1.3

[Luck04] LUCK, Michael; MCBURNEY, Peter; PREIST, Chris. A Manifesto for Agent Technology: Towards Next Generation Computing. **Journal of Autonomous Agents and Multi-Agent Systems**, v. 9, n. 3, pp. 203-253, 2004. 2.2.1, 2.2.5

[Luckschandl01] LUCKSCHANDL, E. Evolving Routing Algorithms with Genetic Programming. **Agent Technology for Communication Infrastructures**, John Wiley & Sons, cap. 22, pp. 287-294, 2001. 5.1

[M.3010] ITU-T Recommendation M.3010. **Principles for a Telecommunications Management Network**, 2000. 2.1.2, 2.1.2

[M.3100] ITU-T Recommendation M.3100. **Generic Network Information Model**, 1995. 2.1.2

[Magedanz96] MAGEDANZ, T.; ROTHERMEL, K.; KRAUSE, S. Intelligent Agents: An Emerging Technology for Next Generation Telecommunications? Em: **Proceedings of the IEEE 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation**, San Francisco, EUA – INFOCOM'96. IEEE, v. 2, pp. 464-472, 1996. 5.1

[MartinFlatin99] MARTIN-FLATIN, Jean-Philippe; ZNATY, Simon; HUBAUX, Jean-Pierre. A Survey of Distributed Enterprise Network and Systems Management Paradigms. **Journal of**

Network and Systems Management, v. 7, n. 1, pp. 9-26, 1999. 2.1.3, 2.1.3

[Menezes96] MENEZES, Alfred; OORSCHOT, Paul; VANSTONE, Scott. **Handbook of Applied Cryptography**. Seção 7.4. CRC, 1996. 2.1.2

[Microsoft] Microsoft Corporation. **The Component Object Model Specification**. 2.1.3

[Minar99] MINAR, Nelson; KRAMER, Kwindla; MAES, Pattie. Cooperating Mobile Agents for Dynamic Network Routing. **Software Agents for Future Communications Systems**, Springer, cap. 12, pp. 287-304, 1999. 5.1

[Morreale98] MORREALE, Patricia. Agents on the Move. **IEEE Spectrum**, v. 35, n. 4, pp. 34-41, 1998. 5.1

[Muller97] MULLER, Nathan. Improving Network Operations with Intelligent Agents. **International Journal of Network Management**, v. 7, n. 3, pp. 116-126, 1997. 5.1

[Neches91] NECHES, Robert; FIKES, Richard; FININ, Tim; GRUBER, Thomas; PATIL, Ramesh; SENATOR, Ted; SWARTOUT, William. Enabling Technology for Knowledge Sharing. **AI Magazine**, v. 12, n. 3, pp. 36-56, 1991. 2.2.5

[Nwana99a] NWANA, Hyacinth; NDUMU, Divine. A Perspective on Software Agents Research. **The Knowledge Engineering Review**, v. 14, n. 2, pp. 125-142, 1999. 2.2.5

[Nwana99b] NWANA, Hyacinth; NDUMU, Divine; LEE, Lyndon; COLLIS, Jaron. ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems. Em: **Proceedings of the Third Annual Conference on Autonomous Agents**, Seattle, EUA. ACM Press, pp. 360-361. 5.3

[Odubiyi01] ODUBIYI, J.; BAYLESS, G.; RUBERTON, E. Victor – Proactive Fault Tracking and Resolution in Broadband Networks Using Collaborative Intelligent Agents. **Agent Technology for Communication Infrastructures**, John Wiley & Sons, cap. 20, pp. 266-274, 2001. 5.3

[OMG04] Object Management Group. **Common Object Request Broker Architecture: Core Specification**. Versão 3.0.3, 2004. 2.1.2, 2.1.3

[OpenWBEM] Open WBEM. Disponível em: http://openwbem.org. 2.1.2

[Papazoglou03] PAPAZOGLOU, M. P.; GEORGAKOPOULOS, D. Service-Oriented Computing. **Communications of the ACM**, v. 46, n. 10, pp. 24-28, 2003. 2.2.1

[Pavlou04] PAVLOU, George; FLEGKAS, Paris; GOUVERIS, Stelios; LIOTTA, Antonio. On Management Technologies and the Potential of Web Services. **IEEE Communications Magazine**, v. 42, n. 7, pp. 58-66, 2004. 2.1.4

[Pierre01] PIERRE, Samuel; GLITHO, Roch. **Proceedings of the Third International Workshop on Mobile Agents for Telecommunications Applications**, Montreal, Canadá – MATA 2001. Springer 2001. 5.1

[PNNI] ATM Forum. **Private Network-Network Interface Specification**. Versão 1.1, 2002. 2.1.1

[Pras95] PRAS, Aiko. **Network Management Architectures**. Tese de Doutorado, University of Twente, Holanda, 1995. 2.1.2

[Pras04] PRAS, Aiko; SCHÖNWÄLDER, Jürgen; FESTOR, Olivier. XML-Based Management of Networks and Services. **IEEE Communications Magazine**, v. 42, n. 7, pp. 56-57, 2004. 2.1.2, 2.1.4

[Q.700] ITU-T Recommendation Q.700. **Introduction to CCITT Signalling System No. 7**, 1993. 2.1.1

[Rahnema93] RAHNEMA, Moe. Overview of the GSM System and Protocol Architecture. **IEEE Communications Magazine**, v. 31, n. 4, pp. 92-100, 1993. 2.1.2

[Raman98] RAMAN, Lakshmi. OSI Systems and Network Management. **IEEE Communications Magazine**, v. 36, n. 3, pp. 46-53, 1998. 2.1.1, 2.1.2

[Redlich98] REDLICH, Jens-Peter; SUZUKI, Masaaki; WEINSTEIN, Stephen. Distributed Object Technology for Networking.

IEEE Communications Magazine, v. 36, n. 10, pp. 100-111, 1998. 2.1.3

[RFC768] RFC 768. POSTEL, Jonathan. **User Datagram Protocol**, 1980. 2.1.2

[RFC791] RFC 791. POSTEL, Jonathan. **Internet Protocol**, 1981. 2.1.1

[RFC854] RFC 854. POSTEL, Jonathan; REYNOLDS, Joyce. **Telnet Protocol Specification**, 1983. 3.3.2

[RFC1052] RFC 1052. CERF, Vinton. **IAB Recommendations for the Development of Internet Network Management Standards**, 1988. 2.1.2

[RFC1157] RFC 1157. CASE, Jeffrey; FEDOR, Mark; SCHOFFSTALL, Martin; DAVIN, James. **A Simple Network Management Protocol**, 1990. 2.1.1, 2.1.2, 2.1.2

[RFC1180] RFC 1180. SOCOLOFSKY, Theodore; KALE, Claudia. **A TCP/IP Tutorial**, 1991. 2.1.2

[RFC1441] RFC 1441. CASE, Jeffrey; MCCLOGHRIE, Keith; ROSE, Marshall; WALDBUSSER, Steven. **Introduction to Version 2 of the Internet-Standard Network Management Framework**, 1993. 2.1.2

[RFC1451] RFC 1451. CASE, Jeffrey; MCCLOGHRIE, Keith; ROSE, Marshall; WALDBUSSER, Steven. **Manager-to-Manager Management Information Base**, 1993. 2.1.2, 2.1.3

[RFC2205] RFC 2205. BRADEN, Robert; ZHANG, Lixia; BERSON, Steven; HERZOG, Shai; JAMIN, Sugih. **Resource Reservation Protocol**, 1997. 5.1

[RFC2328] RFC 2328. MOY, John. **OSPF Version 2**, 1998. 2.1.1

[RFC3165] RFC 3165. LEVI, David; SCHÖNWÄLDER, Jürgen. **Definitions of Managed Objects for the Delegation of Management Scripts**, 2001.

[RFC3411] RFC 3411. HARRINGTON, David; PRESUHM, Randy; WIJNEN, Bert. **An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks**, 2002. 2.1.2, 3.3.2

[RFC3535] RFC 3535. SCHÖNWÄLDER, Jürgen. **Overview of the 2002 IAB Network Management Workshop**, 2003. 2.1.4

[RFC3577] RFC 3577. WALDBUSSER, Steve; COLE, Robert; KALBFLEISCH, Carl; ROMASCANU, Dan. **Introduction to the Remote Monitoring (RMON) Family of MIB Modules**, 2003. 2.1.2

[RFC4251] RFC 4251. YLONEN, Tatu; LONVICK, Chris. **The Secure Shell (SSH) Protocol Architecture**, 2006. 3.3.2

[RFC4271] RFC 4271. REKHTER, Yakov; LI, Tony; HARES, Susan. **A Border Gateway Protocol 4 (BGP-4)**, 2006. 2.1.1

[RFC4741] RFC 4741. ENNS, Rob. **NETCONF Configuration Protocol**, 2006. 2.1.4, 3.3.2

[Rubinstein99] RUBINSTEIN, Marcelo; DUARTE, Otto. Evaluating Tradeoffs of Mobile Agents in Network Management. **Networking and Information Systems Journal**, v. 2, n. 2, pp. 237-252, 1999. 5.1

[Russell06] RUSSELL, Andrew. 'Rough Consensus and Running Code' and the Internet-OSI Standards War. **IEEE Annals of the History of Computing**, v. 28, n. 3, pp. 48-61, 2006. 2.1.2

[Schönwälder00] SCHÖNWÄLDER, Jürgen; QUITTEK, Jüergen; KAPPLER, Cornelia. Building Distributed Management Applications with the IETF Script MIB. **IEEE Journal on Selected Areas in Communications**, v. 18, n. 5, pp. 702-714, 2000. 2.1.3

[Schönwälder03] SCHÖNWÄLDER, Jürgen; PRAS, Aiko; MARTIN-FLATIN, Jean-Philippe. On the Future of Internet Management Technologies. **IEEE Communications Magazine**, v. 41, n. 10, pp. 90-97, 2003. 2.1.4

[Searle69] SEARLE, John. **Speech Acts: An Essay in the Philosophy of Language**. Cambridge University Press, 1969. 2.2.5

[Spring] Spring Application Framework. Disponível em: http://www.springframework.org. 3.5

[Stallings93] STALLINGS, William. **SNMP, SNMPv2, and CMIP. The Practical Guide to Network-Management Standards**. Terceira edição, Addison Wesley, 1993. 2.1.2

[Stallings98a] STALLINGS, William. SNMP and SNMPv2: The Infrastructure for Network Management. **IEEE Communications Magazine**, v. 36, n. 3, pp. 37-43, 1998. 2.1.2, 2.1.2

[Stallings98b] STALLINGS, William. SNMPv3: A Security Enhancement for SNMP. **IEEE Communications Surveys & Tutorials**, v. 1, n. 1, pp. 2-17, 1998. 2.1.2

[Stevenson95] STEVENSON, Douglas. **Network Management: What it is and what it isn't**. 1995. Disponível em: http://www.sce.carleton.ca/netmanage/NetMngmnt/NetMngmnt.html. 1.2

[Strassner03] STRASSNER, John. **Policy-Based Network Management: Solutions for the Next Generation**. Morgan Kaufmann, 2003. 2.1.4

[Sun] Sun Microsystems. **RMI Specification**. Disponível em: http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html. 2.1.3

[Tanaka98] TANAKA, Hiroshi; KOZAI, Shigeo; HORIUCHI, Hiroki; TSUBAKIHARA, Yasunobu; OBANA, Sadao. Architecture of TMN-based Integrated Management System for SDH/PDH Mixed Large-scale Transport Network. Em: **Proceedings of the 1998 IEEE/IFIP Network Operations and Management Symposium**, New Orleans, EUA – NOMS 1998. IEEE Press, v. 3, pp. 815-827, 1998. 2.1.2

[Towle95] TOWLE, Thomas. TMN as Applied to the GSM Network. **IEEE Communications Magazine**, v. 33, n. 3, pp. 68-73, 1995. 2.1.2

[Tunstall02] TUNSTALL, Craig; COLE, Gwyn. **Developing WMI Solutions: A Guide to Windows Management Instrumentation**. Addison-Wesley, 2002. 2.1.2

[Venkateswaran01] VENKATESWARAN, R. Virtual Private Networks. **IEEE Potentials**, v. 20, n. 1, pp. 11-15, 2001. 5.1

[Verma02] VERMA, Dinesh. Simplifying Network Administration Using Policy-Based Management. **IEEE Network**, v. 16, n. 2, pp. 20-26, 2002. 2.1.4

[Voruganti94] VORUGANTI, Ram. A Global Network Management Framework for the '90s. **IEEE Communications Magazine**, v. 32, n. 8, pp. 74-83, 1994. 1.1

[WBEM] Web-Based Enterprise Management. Disponível em: http://www.dmtf.org/standards/wbem. 2.1.2

[WBEMServices] WBEM Services. Disponível em: http://wbemservices.sourceforge.net. 2.1.2

[Wooldridge95] WOOLDRIDGE, Michael; JENNINGS, Nicholas. Intelligent Agents: Theory and Practice. **The Knowledge Engineering Review**, v. 10, n.2, pp. 115-152, 1995. 2.1.2, 2.2.2, 2.2.4

[Wooldridge02] WOOLDRIDGE, Michael. **An Introduction to MultiAgent Systems**. John Wiley & Sons, 2002. 2.2.2

[WS] Web Services Activity Statement, W3C. Disponível em: http://www.w3.org/2002/ws/Activity. 2.1.2, 2.1.4

[WSMAN] Web Services for Management. Disponível em: http://www.dmtf.org/standards/wbem/wsman. 2.1.2

[X.680] ITU-T Recommendation X.680. **Information Technology – Abstract Syntax Notation One (ASN.1): Specification of Basic Notation**, 2002. 2.1.2

[X.703] ITU-T Recommendation X.703. **Information Technology – Open Distributed Management Architecture**, 1997. 2.1.3

[X.710] ITU-T Recommendation X.710. **Information Technology – Open Systems Interconnection – Common Management Information Service**, 1997. 2.1.2

[X.711] ITU-T Recommendation X.711. **Information Technology – Open Systems Interconnection – Common Management Information Protocol: Specification**, 1997. 2.1.1, 2.1.2, 2.1.2

[X.722] ITU-T Recommendation X.722. **Information Technology – Open Systems Interconnection – Structure of Management Information: Guidelines for the Definition of Managed Objects**, 1992. 2.1.2

[X.753] ITU-T Recommendation X.753. **Information Technology – Open Systems Interconnection – Systems Management: Command Sequencer for Systems Management**, 1997. 2.1.3

[XSD] XML Schema, W3C. Disponível em: http://www.w3.org/XML/Schema. 2.1.4

[XML] BRAY, Tim; PAOLI, Jean; SPERBERG-MCQUEEN, Michael; MALER, Eve; YERGEAU, François. **Extensible Markup Language (XML) 1.0**. Quarta edição. World Wide Web Consortium, 2006. 2.1.4

[Ye02] YE, Jian; PAPAVASSILIOU, Symeon; ANASTASI, Giuseppe; PULIAFITO, Antonio. Strategies for Dynamic Management of the QoS of Mobile Users in Wireless Networks through Software Agents. Em: **Proceedings of the 7th International Symposium on Computers and Communications**, Taormina, Itália – ISCC 2002. IEEE, pp. 369-374, 2002. 5.1

[Yemini93] YEMINI, Yechiam. The OSI Network Management Model. **IEEE Communications Magazine**, v. 31, n. 5, pp. 20-29, 1993. 2.1.2

[Yemini00] YEMINI, Yechiam; KONSTANTINOU, Alexander; FLORISSI, Danilo. NESTOR: An Architecture for Network Self-Management and Organization. **IEEE Journal on Selected Areas in Communications**, v. 18, n. 5, pp. 758-766, 2000. 2.1.4

[Zambonelli03] ZAMBONELLI, Franco; PARUNAK, H. Signs of a Revolution in Computer Science and Software Engineering. Em: **Proceedings of the Third International Workshop on Engineering Societies in the Agents World. Lecture Notes in Computer Science**, v. 2577. Springer, pp. 13-28, 2003. 2.2.1

[Zhu01] ZHU, Y.; CHEN, T.; LIU, S. Models and Analysis of Trade-offs in Distributed Network Management Aapproaches. Em: **Proceedings of the 2001 IEEE/IFIP International Symposium on Integrated Network Management**, Seattle, EUA – IM 2001. IEEE, pp. 391-404, 2001. 5.1

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo