Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

Aplicabilidade da Tecnologia JINI na Construção de Sistemas para Gerência de Redes

Autor: Helcio Wagner da Silva
Orientador: Prof. Dr. Luís Geraldo P. Meloni

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica e de Computação.
Área de concentração: **Telecomunicações e Telemática**.

Banca Examinadora

Luís Geraldo P. Meloni, Dr. ..... DECOM/FEEC/UNICAMP
André Leon S. Gradvohl, Dr. ..... Universidade São Francisco/Campus Itatiba
Dalton Soares Arantes, Dr. ..... DECOM/FEEC/UNICAMP
Edmundo Roberto Mauro Madeira, Dr. ..... IC/UNICAMP
Leonardo de Souza Mendes, Dr. ..... DECOM/FEEC/UNICAMP
Maurício Ferreira Magalhães, Dr. ..... DCA/FEEC/UNICAMP

Campinas, SP
Junho/2006

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

Silva, Helcio Wagner da
Si38 Aplicabilidade da tecnologia JINI na construção de sistemas para gerência de redes / Helcio Wagner da Silva. - -Campinas, SP: [s.n.], 2006.

Orientador: Luís Geraldo Pedroso Meloni
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computação - Protocolos. 2. Redes de computação. 3. Sistemas operacionais distribuídas (Computadores). 4. Java (Linguagem de programação de computador). I. Meloni, Luís Geraldo Pedroso. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: Applicability of JINI technology on development of network management systems
Palavras-chave em Inglês: Network management, Computer networks, Distributed Systems, JINI, Java
Área de concentração: Telecomunicações e Telemática
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: André Leon Sampaio, Dalton Soares Arantes, Edmundo Roberto Mauro Madeira, Leonardo de Souza Mendes e Mauricio Ferreira Magalhães
Data da defesa: 14/06/2006

Agradecimentos

Diversas dificuldades surgiram durante a realização deste trabalho, sendo algumas delas consideradas naturais e outras não. À despeito de suas naturezas, pessoas ao longo do caminho contribuíram (quer seja intervindo diretamente ou fornecendo exemplos de conduta) para que elas fossem superadas. Os parágrafos seguintes expõem minhas deferências a essas pessoas. Agradeço, assim:

- a Rogério P. da Costa e Antônio Carlos Lopes Fernandes Jr., não apenas pelas inebriantes aulas de violão (EBGDAE) mas, principalmente, pela prontidão com a qual se dedicaram a "cabrar" minha auto-estima quando ela, por várias vezes, tendia a "picar";

- ao valoroso trabalho de revisão ortográfica e gramatical de meus artigos, realizado pelos irmãos Lucas e Ricardo Batista Freitas (os príncipes da Bahia), por Antônio Marcelo (Max) e por Álvaro Augusto (Gordão);

- a Francisco Helder, Glauco Yared e Manoel Henrique, pelo companheirismo e ajuda mútua diante das inúmeras dificuldades enfrentadas ao longo de nossos respectivos trabalhos;

- a todas as pessoas que contribuíram para a construção de ambientes saudáveis dos quais eu tenha participado ao longo deste doutorado (citá-las em uma única página seria impossível).

Outras pessoas contribuíram para o êxito deste trabalho. Em particular, agradeço a meus pais pelo rigor e pelos vários "nãos" recebidos, os quais me fizeram encarar as dificuldades com a determinação necessária para superá-las. Agradeço também aos meus irmãos, pela confiança que sempre depositaram em mim.

Agradeço, enfim, a Neusa Sales. Com seu carinho, seu amor e seus "sims", esta fantástica mulher inaugurou a fase "Bossa Nova" de minha vida.

"sometimes you can't make it on your own"

(trecho da música homônima, de autoria de
Paul Hewson, o Bono, vocalista da banda irlandesa U2)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

2PC *Two-Phase Commit*

ACID *Atomicity, Consistency, Isolation and Durability*

API *Application Programming Interface*

ARC *Alarm Reporting Control*

BPEL4WS *Business Process Execution Language for Web Services*

CIM *Common Information Model*

CLDC *Connected Limited Device Configuration*

COPS *Common Open Policy Service*

CORBA *Common Object Request Broker Architecture*

DES *Data Encryption Standard*

DISMAN *DIStributed MANagement*

DoS *Denial of Service*

DTD *Document Type Definition*

FTP *File Transfer Protocol*

GCF *Generic Connection Framework*

GDMO *Guidelines for Definition of Managed Objects*

GUI *Graphical User Interfaces*

HTML *Hyper Text Markup Language*

HTTP *HyperText Transfer Protocol*

HTTPMD *HTTP Message Digest*

HTTPS *HTTP over SSL*

IAB *Internet Architecture Board*

IDE *Integrated Development Environment*

IETF *Internet Engineering Task Force*

J2ME *Java 2 Platform Micro Edition*

J2SE *Java 2 Platform Standard Edition*

JAAS *Java Authentication and Authorization Service*

JAR *Java ARchive*

JERI *JINI Extensible Remote Invocation*

JIDM *Joint Inter-Domain Management*

JRMP *Java Remote Method Procotol*

JTSK *JINI Technology Starter Kit*

JVM *Java Virtual Machine*

LDAP *LightWeight Directory Access Protocol*

LDIF *LDAP Data Interchange Format*

LUS *LookUp Service*

M2M *Manager-to-Manager*

MbD *Management by Delegation*

MD5 *Message Digest Number 5*

MHAP *Multicast Host Announcement Protocol*

MHRP *Multicast Host Request Protocol*

MIB *Management Information Base*

MO *Managed Object*

NMS *Network Management System*

OASIS *Organization for the Advancement of Structured Information Standards*

OID *Object IDentifier*

OTS *Object Transaction Service*

PCIM *Policy Core Information Model*

PCLS *Policy Core LDAP Schema*

PDA *Personal Digital Assistant*

PDP *Policy Decision Point*

PDU *Protocol Data Unit*

PEP *Policy Enforcement Point*

PMA *Policy Management Application*

PNMS *Policy-based Network Management System*

QoS *Quality of Service*

RAP *Resource Allocation Protocol*

RMI *Remote Method Invocation*

RMON *Remote MONitoring*

RRP *Registration Request Protocol*

SAML *Security Assertion Markup Language*

SESAME *Secure European System for Applications in a Multi-vendor Environment*

SGMP *Simple Gateway Monitoring Protocol*

SHA *Secure Hash Algorithm*

SMI *Structure of Management Information*

SMON *Switched network MONitoring*

SMP *Simple Management Protocol*

SNMP *Simple Network Management Protocol*

SOAP *Simple Object Access Protocol*

SPKM *Simple Public Key GSS-API Mechanism*

SSL *Secure Socket Layer*

S-SNMP *Secure SNMP*

TCK *Technology Compatibility Kit*

USB *Universal Serial Bus*

USM *User-based Security Model*

VACM *View-based Access Control Model*

W3C *World Wide Web Consortium*

WG *Working Group*

WSDL *Web Services Description Language*

WSN *Web Services Notification*

WSS *Web Services Security*

WTK *Wireless ToolKit*

XACML *eXtensible Access Control Markup Language*

XKMS *XML Key Management Specification*

XML *eXtensible Markup Language*

XMLDS *XML Digital Signature*

Resumo

Apesar das iniciativas surgidas ao longo dos anos no âmbito da IETF, alguns aspectos relacionados à configuração e manutenção dos sistemas desenvolvidos atualmente para a gerência de redes TCP/IP ainda são endereçados de maneira *ad-hoc*. Muitas vezes, é necessária a intervenção humana, principalmente na ocorrência de falhas parciais. Esta Tese apresenta um estudo sobre o uso da tecnologia JINI na construção daqueles sistemas para torná-los mais flexíveis e auto-resilientes.

As considerações gerais feitas neste estudo remetem: à associação de Serviços JINI a sistemas gerenciados; ao agrupamento destes Serviços de Domínios Gerenciados; à incorporação de Interfaces com Usuário flexíveis nos *proxies* para estes Serviços; à utilização dos mecanismos de segurança e de suporte à transações providos por JINI na interação entre Aplicações de Gerência e Serviços JINI; e ao emprego de plataformas com baixo poder computacional.

A fim de validá-las, dois sistemas foram desenvolvidos. O primeiro deles tem seu foco no nível de dispositivo, sendo denominado de Baía de Equipamentos. O outro sistema, mais escalável, tem seu foco na rede como um todo, consistindo na implementação de um PNMS utilizando JINI nas comunicações entre a PMA e o PDP. Neste sistema, SNMPv3 e a MIB de Evento são utilizados para reforço de Políticas junto ao PEP. Um projeto alternativo para este PNMS é também descrito, concebendo o PEP como um Serviço JINI e realizando tal reforço mediante invocação remota de métodos junto àquele componente.

Em seguida, JINI é comparada com outras tecnologias utilizadas para a criação de abordagens revolucionárias (CORBA e Web Services, mais especificamente). Esta comparação, que inclui também a tecnologia utilizada no modelo tradicional, é feita com base na modelagem de aplicações, utilização da infra-estrutura pré-existente, possibilidade de utilização de Interfaces com o Usuário flexíveis e aplicação de tais tecnologias em dispositivos com baixo poder computacional.

O resultado daquela comparação aponta para uma viabilidade equivalente, e às vezes até superior, de JINI frente às demais tecnologias. Esta conclusão deve-se em parte à sua considerável infra-estrutura, que provê suporte, inclusive, ao uso de transações. Além disso, mecanismos auxiliares permitem que usuários de sistemas baseados em JINI gerenciem dispositivos e aplicações através de interfaces flexíveis e personalizadas. E outros destes mecanismos permitem o seu uso em dispositivos com restrições computacionais sem comprometimento de funcionalidade. De fato, estas características garantem a sistemas baseados em JINI um nível de flexibilidade e automação sem precedentes. As direções futuras apontadas remetem à construção de mais sistemas baseados nesta tecnologia, endereçando necessidades específicas e até mesmo absorvendo outras tecnologias.

Abstract

In spite of the initiatives that appeared over the years in context of the IETF, some aspects related to configuration and maintenance of systems currently developed for management of TCP/IP-based networks are still addressed in an ad-hoc manner. Some times, human intervention is needed, mainly during the occurence of parcial failures. This Thesis presents a study on utilization of JINI technology for building of those systems, in order to improve them for more flexibility and self-resilience.

The general considerations performed in this study point to: association of JINI Services to managed systems; grouping them in Managed Domains; incorporation of flexible User Interfaces in their related proxies; utilization of security and transactions mechanisms provided by JINI on their communications with Management Applications; and deployment of both components into low computing power devices.

In order to validate these considerations, two systems were developed. The first one focus on the device level, being named Bay of Devices. The second, more scalable, has its focus on the wide-network level. This latter is an implementation of a PNMS that uses JINI for communication between the PMA and the PDP. It uses SNMP and the Event MIB for enforcement of Policies on PEPs. An alternative design for this PNMS is presented as well, showing the PEP as a JINI Service and performing such enforcement by means of remote method invocations on that component.

Next, JINI is compared to other technologies used in creating revolutionary approaches (more specifically, CORBA and Web Services). That comparison, which includes the technology used on traditional approach as well, was performed taking into account the following parameters: modelling of applications, utilization of existent infrastructure and capabilities related to the utilization of flexible User Interfaces and deployment of low computing power devices in these technologies.

The results emerged from that comparison point to an equivalent, and sometimes better, suitability of JINI when compared to these technologies. This conclusion is due partially to its infrastruture, that also includes provision for using transactions. Moreover, auxiliary mechanisms allow users of JINI-based systems to manage devices and applications by means of flexible and customized User Interfaces. Similar mechanisms allow its deployment on devices with lower computing power without loss of functionality.

In fact, these features provide to JINI-based systems an unpreceeded level of flexibility and automation on their operations. The future directions point to building of additional systems based on this technology, addressing aspects specific to devices and/or applications, and merging other technologies, when possible.

Capítulo 1

Introdução

1.1 A necessidade da gerência de redes padronizada

Definitivamente, as redes de computadores têm experimentado mudanças drásticas em sua estrutura e no seu papel ao longo de sua existência. De ilhas de processamento de dados isoladas, tais redes têm se transformado atualmente em sistemas com alcance global desempenhando uma multiplicidade de tarefas.

Incorporando elementos que vão desde dispositivos com baixo poder computacional até *mainframes* com vasta disponibilidade de recursos computacionais, passando pelos computadores de uso doméstico, as redes de computadores são exploradas ao máximo para tornar a conectividade cada vez mais presente na vida dos seres humanos.

Não obstante, essa evolução de estrutura e papel trouxe consigo custos significantes associados à sua utilização. Por exemplo, falhas em uma rede bancária podem paralisar suas operações; atrasos de propagação em redes empresarias podem resultar em prejuízos financeiros consideráveis; e perda de conectividade em redes hospitalares podem ocasionar diagnósticos imprecisos e até mesmo risco de morte de pacientes.

De forma a monitorar e controlar a operação destas redes, cada vez maiores e mais funcionais, são utilizados sistemas denominados NMSs (*Network Management Systems*). E, de forma a serem o mais adequadamente projetados para endereçar a heterogeidade de dispositivos conectados a estas redes, os NMSs baseiam-se, sempre que possível, em modelos de gerência padronizados.

1.2 Os desafios impostos à gerência atual de redes TCP/IP

A maioria dos NMSs desenvolvidos atualmente para a gerência de redes que utilizam a pilha de protocolos TCP/IP (*Transmission Control Protocol / Internet Protocol*) é baseada em um modelo definido pela IETF (*Internet Engineering Task Force*).

Esse modelo é composto pela especificação de uma sintaxe utilizada para modelagem e armazenamento das informações relevantes à gerência da rede, bem como a definição de um protocolo utilizado para a transferência destas informações entre os componentes dos NMSs.

De concepção simples e bastante adequado aos requisitos demandados pelas redes de outrora, esse modelo foi prontamente utilizado para a construção dos NMSs. No entanto, haja vista a evolução experimentada pelas redes citada na seção anterior, ele foi mostrando-se gradativamente inadequado ao não ser capaz de suprir os novos requisitos impostos.

A fim de supri-los, o modelo sofreu várias mudanças estruturais. Apesar disso, o surgimento de modelos alternativos foi (e vem sendo) incentivado, inclusive em encontros recentes promovidos pelo IAB (*Internet Architecture Board*).

1.3 Escopo e organização desta Tese

Dentre os modelos alternativos propostos para gerência de redes TCP/IP, figuram iniciativas relacionadas à utilização de tecnologias originalmente desenvolvidas para a construção de sistemas distribuídos de âmbito geral.

Apesar do surgimento destas iniciativas no desenvolvimento de NMSs que as utilizam, alguns pontos permanecem em aberto, mormente aqueles relacionados à configuração e manutenção dos principais componentes daqueles NMSs.

Esta Tese introduz e investiga a utilização de JINI, uma tecnologia também criada para o desenvolvimento de sistemas distribuídos de âmbito geral, na construção de NMSs para gerência de redes TCP/IP.

A despeito do fato de ser esta uma proposta que segue a mesma filosofia de outras, características intrínsecas à tecnologia investigada, somadas a um estudo sobre sua eficiente utilização na construção de NMSs, garantem flexibilidade e auto-resiliência àqueles sistemas.

De forma a melhor descrever este estudo, o próximo capítulo desta Tese aborda algumas das iniciativas surgidas no âmbito da IETF para a gerência de redes TCP/IP. Particularmente, dá-se ênfase às iniciativas posteriormente mencionadas ao longo da Tese.

Em seguida, o Capítulo 3 expõe os conceitos básicos acerca da tecnologia JINI. Isto é feito inicialmente com um breve histórico desta tecnologia, logo após abordando seus principais aspectos arquiteturais e de comunicação. Por último, é descrito seu modelo de segurança, disponível nas suas versões mais recentes.

O Capítulo 4 apresenta as considerações gerais que devem ser levadas em conta para a construção de NMSs utilizando JINI. Os Capítulos 5 e 6 descrevem dois NMSs que funcionam como provas de conceitos para tais considerações.

De modo geral, para cada um destes NMSs é fornecida uma visão geral sobre sua arquitetura, a partir da qual procede-se a uma análise mais pormenorizada de seus componentes. Essa análise é sucedida por um exemplo de operação do NMS, de forma a ilustrar as interações definidas entre tais componentes.

No Capítulo 7, outras tecnologias voltadas à construção de sistemas distribuídos são apresentadas. Algumas das abordagens criadas para gerência de redes que utilizam aquelas tecnologias também são descritas. Em seguida, é realizada uma comparação entre JINI e aquelas tecnologias. Essa comparação, que inclui também a tecnologia utilizada no modelo tradicional, é feita com base nos seguintes parâmetros: modelagem de Serviços, utilização da infra-estrutura, possibilidade de emprego de Interfaces com o Usuário flexíveis e aplicabilidade em dispositivos com baixo poder computacional.

O Capítulo 8 finaliza a Tese, descrevendo os objetivos alcançados a partir dela e apontando direções para o surgimento de trabalhos futuros.

Capítulo 2

Recentes avanços em Gerência de Redes surgidos no âmbito da IETF

2.1 Introdução

O paradigma tradicionalmente utilizado na gerência de redes TCP/IP envolve a modelagem de informações em MOs (*Managed Objects*), o armazenamento desses MOs em MIBs (*Management Information Bases*) e a transferência dessas informações entre aplicações, denominadas Gerente e Agente, por meio do protocolo SNMP (*Simple Network Management Protocol*).

Este capítulo descreve, de forma sumarizada, algumas das principais iniciativas surgidas ao longo dos anos no contexto da IETF para flexibilizar e distribuir a atividade de gerência de redes TCP/IP. Essas iniciativas estão relacionadas à evolução do protocolo utilizado na transferência de MOs, ao projeto de MIBs específicas, ou mesmo a mudanças aparentemente radicais no paradigma tradicionalmente adotado.

As iniciativa citadas no parágrafo anterior são classificadas em [Schönwälder et al., 2003] como *abordagens evolucionárias*, no sentido em que elas são entendidas como sendo melhoramentos do paradigma utilizado. Iniciativas pertencentes a essa classe encontram-se cobertas em seções específicas deste capítulo. Em particular, um histórico evolutivo sumarizado do protocolo de gerência tradicionalmente utilizado é descrito na Seção 2.2. A Seção 2.3 aborda algumas MIBs voltadas à descentralização da atividade de gerência, e a Seção 2.4 descreve o paradigma de Gerência Baseada em Políticas. Já a Seção 2.5 efetua as considerações finais acerca da revisão realizada neste Capítulo.

2.2 Iniciativas relacionadas ao projeto do protocolo

Atualmente, a gerência padronizada de redes TCP/IP é realizada com o auxílio do protocolo SNMP. Esse protocolo tem experimentado uma série de evoluções ao longo do tempo e, a fim de melhor descrevê-las, os comentários a seguir tomam como base a Figura 2.1.

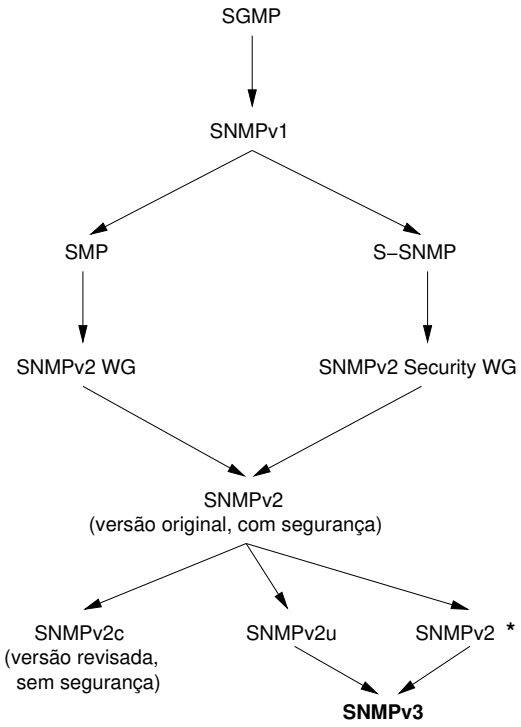

Figura 2.1: Evolução do SNMP.

O ponto de partida para o SNMP foi o SGMP (*Simple Gateway Monitoring Protocol*), especificado em [Davin et al., 1987] e utilizado primordialmente para a monitoração de *gateways*. Quando foi identificada a necessidade de uma ferramenta de caráter mais geral para gerência de redes, o SGMP foi tomado como base e adaptado para produzir aquela que é a primeira versão do protocolo SNMP, conhecida como SNMPv1.

O conjunto de especificações SNMPv1 definia mais que apenas um protocolo [Case et al., 1990]. Ele definia também um modelo para a formatação e armazenamento da informação de gerência em MIBs, denominada SMI (*Structure of Management Information*) [Rose e McCloghrie, 1990], e um conjunto de MOs de propósito geral conhecido como MIB-II [McCloghrie e Rose, 1991].

Essa primeira versão do protocolo obteve uma grande aceitação de mercado, principalmente devido à sua facilidade de compreensão e implementação. Porém, o protocolo era considerado inflexível haja vista que fazia parte de um modelo fortemente centralizado em torno de uma única estação de gerência.

Outro problema do SNMPv1 era a falta de suporte à transferência volumosa de dados, necessária

principalmente quando tabelas de MOs eram coletadas. Por fim, a especificação carecia de características de segurança que proibissem o mascaramento, a modificação e a observação do conteúdo das mensagens trocadas.

Para remediar os problemas relacionados à segurança, um conjunto de documentos referenciado como S-SNMP (*Secure SNMP*) foi emitido como *proposed standards* em Julho de 1992. Naquele mesmo mês, o SMP (*Simple Management Protocol*) foi emitido fora da estrutura de padrões da Internet, provendo melhorias funcionais ao SNMPv1 e incorporando as melhorias de segurança propostas no S-SNMP.

Para desenvolvimento da segunda geração do SNMP, dois Grupos de Trabalho foram formados. O SNMPv2 WG (*Working Group*) tomou como base o SMP, sendo encarregado dos aspectos funcionais do protocolo. O segundo Grupo de Trabalho era conhecido como SNMPv2 *Security* WG e, tomando como base o S-SNMP, foi encarregado dos aspectos de segurança do protocolo. Esse esforço combinado deu origem então ao SNMPv2, um protocolo que acumulava melhorias funcionais e de segurança sobre o SNMPv1.

No entanto, após anos de utilização do SNMPv2 a IETF fez com que um novo Grupo de Trabalho revisasse as especificações e, no novo conjunto de documentos emitidos, as características de segurança originalmente propostas foram removidas. Ao invés disso, o SNMPv2 fez uso do mesmo mecanismo de segurança proposto para o SNMPv1, o que levou a comunidade a denominá-lo SNMPv2c (de *community-based*, uma alusão ao mecanismo de segurança presente no SNMPv1).

Descrito em [Case et al., 1996], o SNMPv2c provia um modelo mais descentralizado através da criação de Gerentes de nível intermediário e de um novo tipo de PDU (*Protocol Data Unit*) especificamente projetado para a comunicação com estes Gerentes. Semelhantemente, um novo tipo de PDU foi especificamente projetado para a transferência volumosa de dados.

A despeito das funcionalidades providas pelo SNMPv2c, essa segunda versão do protocolo não obteve sucesso semelhante ao alcançado pela versão anterior, principalmente pelo fato de que não possuia mecanismos de segurança eficazes.

Para remediar essa situação, vários Grupos de Trabalho independentes começaram a trabalhar em melhorias de segurança para o SNMPv2. Dentre estes Grupos, dois destacaram-se: o SNMPv2u e o SNMP2*. Os trabalhos realizados por esses dois Grupos serviram como motivação para um novo Grupo de Trabalho da IETF, denominado SNMPv3 WG. Este Grupo foi o responsável pela criação da terceira e atual versão do protocolo, que encontra-se especificado em [Harrington et al., 1998].

O SNMPv3 absorve as funcionalidades surgidas no SNMPv2c e possui ainda as tão desejadas melhorias de segurança. Em particular, seu modelo de segurança, denominado USM (*User-based Security Model*) [Blumenthal e Wijnen, 1998], permite que autenticação seja empregada mediante utilização dos protocolos MD5 (*Message Digest Number 5*) [Rivest, 1992] ou SHA (*Secure Hash Algorithm*) [NIST, 1995]. Esse modelo também é capaz de prover confidencialidade, mediante utilização do protocolo DES (*Data Encryption Standard*) [ANSI, 1983]. O SNMPv3 conta ainda com um mecanismo denominado VACM (*View-based Access Control Model*) [Wijnen et al., 1998] para acesso granularizado aos MOs presentes nas MIBs suportadas pelos Agentes.

2.3 Iniciativas relacionadas ao projeto de novas MIBs

Algumas outras iniciativas surgiram no intuito de prover maior flexibilidade e distribuição aos NMSs (*Network Management Systems*). No entanto, essas últimas iniciativas foram baseadas não em alterações no protocolo utilizado, e sim no projeto de MIBs específicas. A primeira dessas iniciativas pode ser encontrada num dos documentos sob responsabilidade do outrora formado SNMPv2 WG. Esse documento [Case et al., 1993] define uma MIB M2M (*Manager-to-Manager*) que facilita a delegação de tarefas relacionadas a *polling* para Gerentes de nível intermediário. No entanto, haja vista que essa MIB pode ser utilizada apenas para tarefas de gerência simples, ela é geralmente vista como um ponto de partida para iniciativas desta natureza, possuindo atualmente o *status* de *historic* por parte da IETF.

Um outro trabalho pioneiro nesta área foi realizado pelo *RMON MIB WG* [IETF, 2005b]. Esse Grupo de Trabalho é responsável pela MIB RMON (*Remote MONitoring*) [Waldbusser, 2000]. Essa MIB define MOs para uso em dispositivos remotos, denominados *sondas* ou *monitores*, localizados em segmentos de rede e responsáveis por coletar informações sobre estatísticas de tráfego nestes segmentos.

Embora a maioria dos MOs presentes na MIB RMON permita gerenciar qualquer tipo de rede, alguns deles são particularmente destinados à gerência de redes Ethernet. Por outro lado, a MIB RMON Token Ring, especificada em [Waldbusser, 1993], provê MOs específicos à gerência de redes daquele tipo. Já a MIB RMON-2, especificada em [Waldbusser, 1997], estende a MIB RMON à aplicação em protocolos outros que aqueles de enlace e de rede, o que era o foco da MIB RMON original. Por fim, a MIB SMON (*Switched network MONitoring*), especificada em [Waterman et al., 1999], estende a MIB RMON de forma a prover a gerência de redes comutadas.

Atualmente, a definição de MOs utilizados para a gerência distribuída está a cargo do DISMAN (*DIStributed MANagement*) WG [IETF, 2005a]. Este Grupo de Trabalho está atuando em três abordagens diferentes. A primeira delas pode ser vista como um melhoramento da MIB M2M, sendo representada pelas MIBs de Expressão, de Evento, de *Log* de Notificação, de Alarme, e a MIB ARC (*Alarm Reporting Control*). Já a segunda abordagem é baseada na utilização de *scripts*, sendo representada pelas MIBs *Script* e *Scheduling*. Por fim, a terceira abordagem tem seu foco numa MIB para funções de gerência específicas que podem ser invocadas em dispositivos remotos.

A MIB de Expressão, especificada em [Kavasseri e Stewart, 2000a], suporta expressões definidas externamente envolvendo MOs pré-existentes. Estas expressões constituem-se também em MOs, e os resultados de suas respectivas avaliações são disponíveis na MIB assim como os valores de quaisquer outros MOs. A MIB de Evento, especificada em [Kavasseri e Stewart, 2000b], provê a habilidade para monitorar MOs em sistemas locais ou remotos, tomando medidas simples quando determinadas condições ocorrem. Essa MIB foi projetada com base nas experiências de projeto, implementação e utilização da MIB RMON, e é destinada preferencialmente à operação em conjunto com a MIB de Expressão.

A MIB de *Log* de Notificação, especificada em [Kavasseri e Stewart, 2000c], provê uma função de *log* para emissores de notificações, sejam elas representadas por PDUs *Traps* ou *Informs*, que excedem o limite máximo de retransmissão. Através dessa MIB, aplicações Gerentes podem efetuar a leitura de seus MOs e obter informações que outrora estariam indisponíveis. A MIB de Alar-

me, especificada em [Chisholm e Romascanu, 2004], define MOs utilizados para a modelagem e o armazenamento de alarmes. Por fim, a MIB ARC (*Alarm Reporting Control*), especificada em [Lam et al., 2004], define MOs utilizados para controlar o reporte de condições de alarme.

A MIB *Script*, especificada em [Levi e Schöenwäelder, 2001], permite a transferência de *scripts* de gerência para localizações remotas de forma a criar e controlar a execução de *threads* de gerência. Trata-se, portanto, de uma abordagem que visa o emprego do conceito de MbD (*Management by Delegation*), exposto inicialmente em [Yemini et al., 1991] e [Goldszmidt e Yemini, 1995], por parte da IETF. A MIB *Scheduling*, especificada em [Levi e Schöenwäelder, 2002], possibilita o agendamento de ações periodicamente ou em momentos específicos. Esta MIB é projetada para agendar a execução de *scripts* na MIB *Script*.

A MIB de Operações Remotas, definida em [White, 2000], define módulos destinados à execução de determinadas operações remotas, mais especificamente as operações de `ping`, `traceroute` e funções de *lookup*. De acordo com esta MIB, um *host* local inicializa a execução destas operações junto a um *host* alvo mediante o envio de uma solicitação SNMP para um *host* intermediário, denominado *host* remoto. Neste *host* remoto, MOs são configurados e, em consequência disto, as operações são realizadas de fato pelo *host* remoto tendo como destino o *host* alvo.

As abordagens nas quais atua o DISMAN WG não devem ser compreendidas como competidoras, mas sim como complementares entre si. Para uma determinada tarefa pode ser melhor utilizar uma abordagem, e para outra tarefa outra abordagem. Infelizmente, não é fácil integrar as três abordagens e utilizar, por exemplo, a MIB *Scheduling* em combinação com a MIB de Evento.

Em uma das implementações descritas nesta Tese, é utilizada a MIB de Evento. Essa MIB possibilita a monitoração de MOs e a definição de eventos que ocorrem quando gatilhos (*triggers*) são disparados com base em testes realizados naquele MO. Esses eventos podem resultar na configuração de MOs na MIB suportada pelo Agente e no envio de Notificações SNMP para um Gerente.

A Figura 2.2, extraída de [Kavasseri e Stewart, 2000b], fornece uma visão geral das nove tabelas da MIB de Evento e dos relacionamentos existentes entre elas.

Para criar um gatilho, é necessário criar uma linha na tabela `mteTriggerTable`. Nessa linha, o MO `mteTriggerValueID` especifica o OID (*Object IDentifier*) do MO a ser observado, e o MO `mteTriggerValueIDWildcard` especifica se este OID refere-se a uma única instância do MO, ou a múltiplas instâncias. Neste momento, uma linha na tabela `mteObjectsTable` é criada contendo o OID definido no MO `mteTriggerValueID`.

Logo após, naquela mesma tabela, é configurado o MO `mteTriggerTest`, que define o tipo de teste a ser realizado sobre o valor do MO. Em particular, esse teste pode ser um teste de existência, de valor-limite (*threshold*), ou um teste do tipo *boolean*. Em seguida, é configurado o MO `mteTriggerFrequency` para a definição do tempo de amostragem do MO monitorado, e configurado o MO `mteTriggerSampleType` para estabelecer se o teste deve ser aplicado tomando como base o valor de apenas uma amostra daquele MO, ou a diferença entre os valores de amostras consecutivas.

Na tabela `mteEventTable`, é criada uma linha relativa ao tipo de ação a ser realizada quando o gatilho é disparado. Conforme comentado anteriormente, isso pode incluir a configuração de MOs

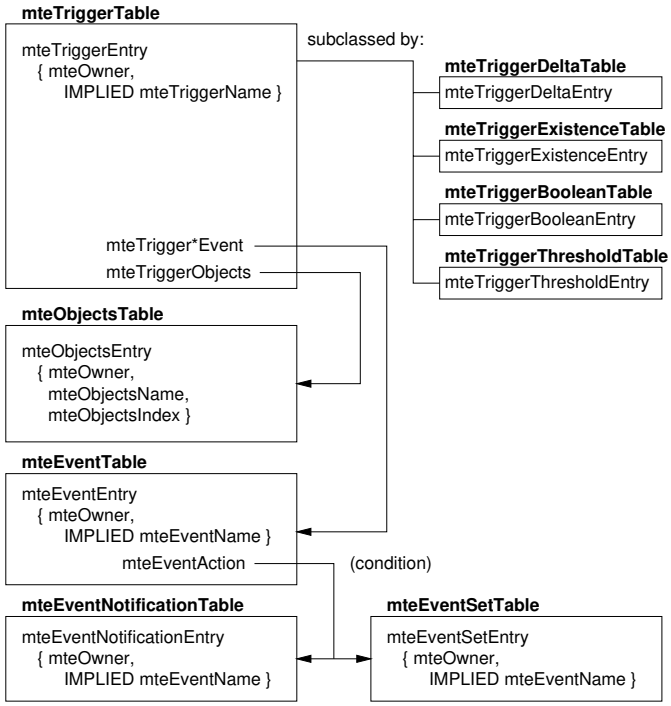

Figura 2.2: Visão geral das tabelas presentes na MIB de Evento.

e o envio de Notificações SNMP para um determinado *host*. A depender da opção escolhida, pode ser utilizada a tabela `mteEventNotificationTable`, que contém uma lista de MOs correspondente à linha da tabela `mteEventTable`. Essa lista de MOs, juntamente com seus respectivos valores, é adicionada à notificação SNMP enviada quando o gatilho for disparado. Uma outra opção é a tabela `mteEventSetTable`, que pode conter um MO para cada evento definido na tabela `mteEventTable`. Quando um gatilho for disparado, o MO é configurado no valor especificado por aquela tabela.

Uma vez configurados os parâmetros gerais do gatilho e as ações a serem levadas a cabo na ocorrência de um evento disparado por aquele gatilho, parte-se então para a definição de seus parâmetros específicos. Isso porque, dependendo do tipo de teste a ser realizado com o MO monitorado, o gatilho recebe a denominação de gatilho de existência, de valor-limite ou gatilho do tipo *boolean*. O resultado disso é a configuração de MOs em uma das seguintes tabelas: `mteTriggerDeltaTable`, `mteTriggerExistenceTable`, `mteTriggerThresholdTable` ou `mteTriggerBooleanTable`.

2.4 Gerência Baseada em Políticas

Uma das mais recentes metodologias criadas pela IETF para a gerência de redes é denominada *Gerência Baseada em Políticas*, cujos pilares encontram-se especificados em [Yavatkar et al., 2000].

Apesar de surgida num contexto da gerência do controle de admissão aos recursos de rede, essa metodologia pode seguramente ser aplicada em outros problemas de gerência mais gerais.

De acordo com [Westerinen et al., 2001], que define a terminologia utilizada na Gerência Baseada em Políticas, uma *Política* pode ser vista de duas perspectivas: um objetivo definido, um curso ou método de ação para guia e determinação de decisões presentes e futuras ou um conjunto de ações para administrar, gerenciar e controlar o acesso aos recursos de rede.

Uma Política pode ser representada em níveis diferentes, indo de objetivos de negócios até parâmetros de configuração. Daí falar-se em *Políticas de alto nível*, especificadas num formato que é intuitivo aos administradores de rede, e *Políticas de baixo nível*, precisamente e consistentemente especificadas para cada equipamento na rede.

Políticas podem ser representadas por meio do PCIM (*Policy Core Information Model*), especificado em [Moore et al., 2001]. O PCIM é uma extensão do CIM (*Common Information Model*) [DMTF, 2005], que provê definições semanticamente ricas de informações de gerência para sistemas, redes, aplicações e serviços, permitindo ainda definições específicas de fabricantes.

A Figura 2.3 provê uma visão geral das Classes básicas do PCIM e de seus relacionamentos. Conforme observado, seu bloco de construção básico é a *Regra de Política*, representada pela Classe `PolicyRule`. Ela é a ligação de um conjunto de *Ações de Política*, representadas pela Classe `PolicyAction`, a um conjunto de *Condições de Política*, representadas pela Classe `PolicyCondition`. As Condições são avaliadas para determinar se as Ações são ou não realizadas.

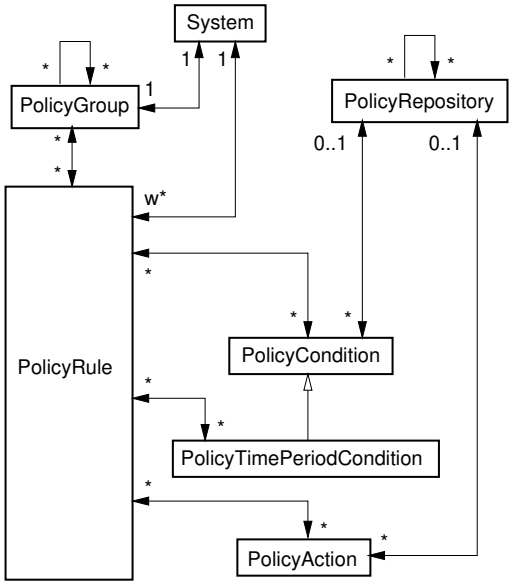

Figura 2.3: Visão geral das Classes básicas do PCIM e de seus relacionamentos.

Condições e Ações de Política podem ser reutilizáveis ou específicas a uma Regra de Política, sendo armazenadas num *Repositório de Políticas*, representado na Figura 2.3 pela Classe `PolicyRepository`. Regras de Políticas podem ser combinadas num *Grupo de Políticas*, representado na

Figura 2.3 pela Classe `PolicyGroup`, e Grupos de Política podem conter outros Grupos de Políticas.

A Figura 2.4 ilustra os principais componentes que fazem parte de um PNMS (*Policy-based Network Management System*).

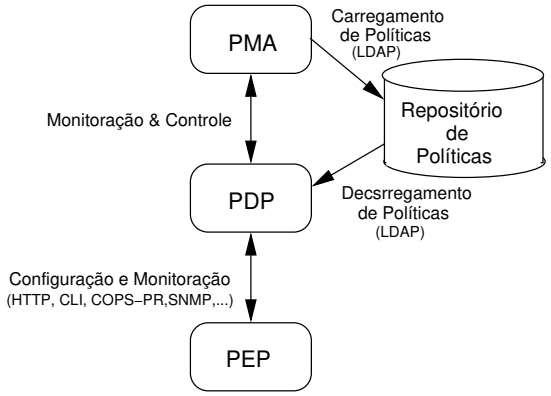

Figura 2.4: Principais componentes de um PNMS.

A PMA (*Policy Management Application*) provê a interface para administradores de rede criarem e empregarem Políticas, armazenando-as num Repositório de Políticas, e monitorar o estado do ambiente gerenciado por Políticas.

O PDP (*Policy Decision Point*) é uma entidade lógica que extrai as Regras de Política do Repositório e toma decisões por ele próprio ou por outros elementos de rede que solicitam tais decisões. Essas decisões são representadas pela avaliação do conjunto de condições das Regras de Política.

O PEP (*Policy Enforcement Point*) é uma entidade lógica que reforça decisões de Políticas. Esse reforço é representado pela execução do conjunto de ações das Regras de Política. Em particular, essa execução é solicitada pelo PDP quando o conjunto de condições associado a uma Regra de Política é avaliado em `TRUE`.

Uma sugestão para protocolo de acesso ao Repositório de Políticas que vem tendo uma razoável aceitação é o LDAP (*Lightweight Directory Access Protocol*), especificado em [Wahl et al., 1997]. No LDAP as informações são armazenadas num Servidor de acordo com uma estrutura de diretório/árvore, sendo acessíveis a um ou mais Clientes. A IETF define em [Strassner et al., 2004] um mapeamento do PCIM para um *schema* LDAP, denominado PCLS (*Policy Core LDAP Schema*). O PCLS define duas hierarquias de Classes de Objeto: *Classes estruturais*, que contêm informações para representar e controlar dados de Políticas como representados em [Moore et al., 2001], e *Classes de relacionamento* que indicam quantas instâncias das Classes estruturais estão relacionadas umas às outras.

Uma sugestão para protocolo de transferência de decisões de Políticas entre PDPs e PEPs é o protocolo COPS (*Common Open Policy Service*). Definido pelo RAP (*Resource Allocation Protocol*) WG [IETF, 2004] e especificado em [Durham et al., 2000], o COPS descreve um modelo Cliente/Servidor simples criado para suportar controle de Políticas sobre protocolos de sinalização de QoS (*Quality of Service*). Nele, o PEP, movido por uma entidade externa, contacta o PDP com

vistas a obtenção de uma decisão de Políticas. Esse modo de operação é conhecido também como *outsourcing*.

Outro modo de operação para o protocolo COPS é denominado de *provisionamento de políticas*, especificado em [Chan et al., 2001]. Nesse modo o protocolo, denominado então COPS-PR (*PRovisioning*), é orientado a eventos. Em outras palavras, não há *polling* entre PEPs e PDPs; ao invés disso, o PDP, movido por uma entidade externa, envia dados ao PEP.

Outros protocolos também podem ser utilizados para transferir decisões de Políticas entre PDPs e PEPs, tais como HTTP (*HyperText Transfer Protocol*) [Fielding et al., 1999], FTP (*File Transfer Protocol*) [Postel e Reynolds, 1985] ou o próprio SNMP, essa última opção sendo adotada em uma das implementações descritas nesta Tese.

2.5 Conclusão

Este capítulo descreveu algumas das recentes evoluções surgidas no âmbito da IETF para a gerência de redes TCP/IP. Essas evoluções correspondem a melhorias no protocolo SNMP, elaboração de novas MIBs e criação do paradigma de Gerência Baseada em Políticas.

Embora descreva apenas parte do conjunto total das abordagens evolucionárias surgidas ao longo dos anos, este capítulo harmoniza-se com os demais subseqüentes pela introdução de algumas das tecnologias empregadas nos dois protótipos a serem descritos nesta Tese. Em particular, ambos utilizam SNMPv3, em maior ou menor extensão. E, conforme será descrito posteriormente, no projeto de um deles são combinados o paradigma de Gerência Baseada em Políticas, SNMPv3 e a MIB de Evento.

Capítulo 3

Tecnologia JINI: uma visão geral

3.1 Introdução

O objetivo deste Capítulo é descrever os principais elementos que compõem a tecnologia JINI, e a forma como eles correlacionam-se de maneira a prover os benefícios colhidos a partir da utilização dessa tecnologia.

Para tal, a Seção 3.2 descreve um breve histórico acerca de JINI. Já a Seção 3.3 descreve seus principais aspectos arquiteturais e de comunicação, enquanto a Seção 3.4 aborda os principais aspectos de segurança introduzidos nas mais recentes versões desta tecnologia. A Seção 3.5 apresenta um resumo desta exposição.

3.2 Um breve histórico

Em dezembro de 1990 surgiu na Sun Microsystems o projeto *Green*, capitaneado por Patrick Naughton, Mike Sheridan e James Gosling. Seu objetivo era investigar quais seriam as novas tendências da computação, e como melhor tirar proveito delas. Decidiu-se que pelo menos uma delas seria a convergência entre equipamentos eletrônicos e computadores, ou seja, sistemas embarcados. Um dos produtos desse projeto foi *Oak*, uma linguagem portável destinada à construção de aplicações para execução naqueles sistemas.

Em abril de 1995, a linguagem foi rebatizada de *Java* e lançada conjuntamente com o navegador Web *HotJava*. De fato, muitos dos conceitos básicos por trás da linguagem Java e que surgiram a partir de sua aplicabilidade em equipamentos eletrônicos (independência de arquitetura de CPU, segurança, compactação) foram bem recebidas na sua nova casa: a Web.

Apesar do sucesso obtido por Java em ambientes de aplicação *desktop*, a visão original dos

engenheiros da Sun acerca de um mundo no qual dispositivos eletrônicos possam comunicar-se entre si de maneira fácil e independente não morrera. Mark Weiser, da Xerox PARC, chamou essa visão de *computação ubíqua*, um termo significante para expressar a pronta disponibilidade e usabilidade de dispositivos em relação à rede. Apesar de Java tornar possível a existência desse mundo, mecanismos adicionais deveriam ser projetados de forma a facilitar a sua criação.

Com a visão de computação ubíqua em mente, os engenheiros da Sun concentraram-se na tarefa de construir uma infra-estrutura de software que utilizasse os serviços providos por Java e provesse os benefícios de confiabilidade, manutenibilidade, escalabilidade e espontaneidade que aquela visão requeria.

Esse projeto tornou-se conhecido como *JINI*, e muitas das pessoas que o desenvolveram foram as mesmas que desenvolveram originalmente alguns aspectos de Java. Bill Joy tornou-se um dos chefes do projeto, exatamente como ocorrera originalmente no projeto Green. Jim Waldo tornou-se o arquiteto e projetista chefe do projeto. Ann Wollrath desenvolveu a Java RMI (*Remote Method Invocation*), e continuou o seu trabalho no contexto de JINI. Ken Arnold desenvolveu os modelos de transações e armazenamento em JINI. Bob Scheifler, o único dos envolvidos que não era filiado à Sun, definiu os protocolos de consulta e descoberta.

Há um número de modelos exóticos de programação que poderiam servir de base para o desenvolvimento de JINI. No entanto, seus desenvolvedores preferiram usar como base os benefícios providos pela linguagem Java, e a partir daí definir a infra-estrutura que provê os conceitos necessários à concretização da visão de computação ubíqua.

O projeto JINI esteve longe dos olhos do público até que o repórter John Markoff, do New York Times, descobriu a história e a publicou num artigo em 1998. Brevemente após, JINI apareceu na capa da revista *Wired*, embora não tinha sido ainda formalmente anunciada. Na verdade, esse anúncio formal só veio a ser feito em 25 de janeiro de 1999. Atualmente, JINI encontra-se na sua versão 2.1, e possui uma comunidade ativa de desenvolvedores e usuários que promove a sua difusão mundo afora. Essa comunidade, a *Comunidade JINI* (`http://www.jini.org`), permite o surgimento de novas iniciativas que contemplam desde melhorias estruturais na tecnologia até mecanismos endereçando problemas específicos. Essas iniciativas tomam a forma de *Projetos* surgidos no âmbito da Comunidade JINI.

3.3 Aspectos básicos de arquitetura e comunicação

JINI é uma tecnologia criada para a construção de sistemas distribuídos, sendo fortemente baseada na linguagem de programação Java. De acordo com [Sun Microsystems, 2003], sistemas construídos com base nessa tecnologia podem ser arquiteturalmente desmembrados em três categorias, a saber:

- um conjunto de componentes que provêem uma infra-estrutura para federar *Serviços* num sistema distribuído;

- um modelo de programação que suporta e encoraja a produção de sistemas distribuídos confiáveis;
- Serviços que aderem e retiram-se de um sistema JINI federado e que oferecem funcionalidades a Clientes e outros Serviços federados.

Embora essas categorias sejam separáveis e distintas, elas relacionam-se entre si. De fato, o modelo de programação, que consiste numa API (*Application Programming Interface*), é utilizado pelos Serviços federados e pelos componentes da infra-estrutura, sendo suportado por esses últimos.

Os objetivos finais de um sistema construído a partir da tecnologia JINI encontram-se sumarizados abaixo:

- permitir que Clientes compartilhem Serviços numa infra-estrutura de rede;
- permitir que esses Clientes acessem facilmente os Serviços localizados em qualquer parte da infra-estrutura, mesmo que a localização desses Clientes dentro da infra-estrutura varie livremente;
- simplificar a tarefa de desenvolver e, principalmente, efetuar a manutenção desse sistema, independentemente de qual seja seu tamanho.

Serviços JINI são entidades computacionais que implementam uma ou mais *interfaces* publicamente conhecidas. Esses Serviços agrupam-se em torno de uma ou mais *Comunidades*, mediante interação com um Serviço denominado LUS (*LookUp Service*). O conjunto de todas as Comunidades, por sua vez, forma uma *Federação*.

A interação entre Clientes e Serviços JINI se dá mediante a utilização de *proxies* para estes Serviços. A maneira segundo a qual estes *proxies* são obtidos pelos Clientes faz parte do mecanismo de comunicação utilizado em JINI. Em particular, esse mecanismo faz uso intensivo de características-chave presentes na tecnologia Java, como o descarregamento dinâmico de Classes a partir da rede, a serialização de Objetos e a anotação de *codebases*.

A Figura 3.1 ilustra alguns detalhes deste mecanismo. Nessa figura, um determinado Serviço adere a uma Comunidade registrando seu *proxy* serializado junto ao LUS (a). Em seguida, um Cliente contacta o LUS e obtém este *proxy* serializado, que contém a anotação do *codebase* para seu respectivo código (b). De posse dessa informação, o Cliente contacta um Servidor de *codebase* e descarrega o código referente ao *proxy* (c). Esse Cliente pode então instanciar o *proxy* e invocar operações junto ao mesmo (d), resultando na comunicação com o Serviço.

A implementação de referência provida pela Sun Microsystems para um LUS denomina-se `reggie`. Conforme exposto, esse Serviço funciona como um *meta Serviço*, provendo acesso aos demais Serviços. Os protocolos utilizados por um Serviço JINI para descobri-lo e registrar com ele seu *proxy* serializado são denominados de protocolos de *descoberta* e *adesão*, respectivamente. Em particular, dois protocolos de descoberta podem ser empregados: um *unicast* e outro *multicast*.

Serviços JINI podem ser encontrados com base em três parâmetros: um identificador único, uma ou mais interfaces implementadas e um conjunto de atributos do Serviço.

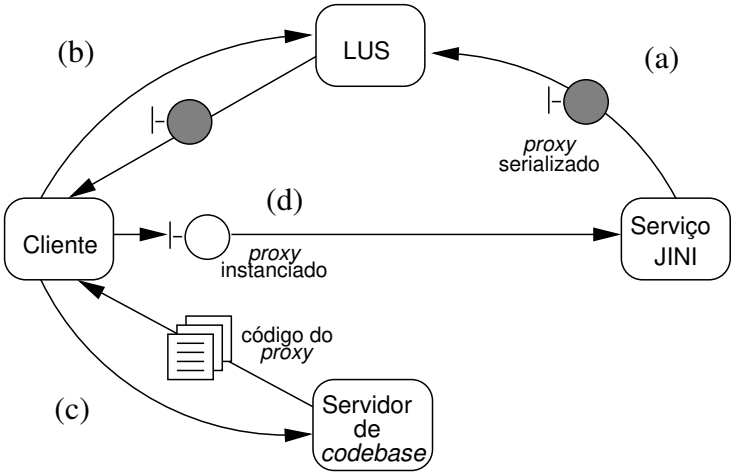

Figura 3.1: Mecanismo de comunicação utilizado em JINI.

Embora o LUS atribua a cada um dos Serviços registrados um identificador único, um potencial Cliente do Serviço não conhece este identificador a menos que já tenha entrado em contato inicialmente com ele. Assim, a busca por identificador não é a melhor forma de se contactar um determinado Serviço, pelo menos não pela primeira vez. A melhor forma de realizar este contato inicial é através do conhecimento prévio da interface implementada pelo Serviço. Esta interface é muitas vezes vista como o contrato entre um Cliente e um Serviço JINI.

O registro de *proxies* serializados junto ao LUS é mantido segundo uma disciplina de *leasing*. Ou seja, ao fim de cada registro é obtido um *lease* que deve ser renovado periodicamente para que o LUS mantenha o *proxy* associado ao Serviço.

A adoção de uma política de *lease* tem por fim minimizar os efeitos de falhas parciais e possibilitar a auto-configuração da Comunidade. Mais especificamente, quando a execução de um determinado Serviço é abruptamente ou suavemente interrompida, o *lease* associado ao registro de seu *proxy* serializado não é mais renovado. Como consequência, o *proxy* é removido do LUS, não estando mais disponível a qualquer Cliente em potencial. Isso elimina a necessidade de administração humana para remover registros antigos.

Um dos aspectos importantes da tecnologia JINI é a sua neutralidade de protocolo. Em outras palavras, não há nenhuma definição estrita do protocolo a ser utilizado na comunicação entre Clientes e Serviços JINI. Isso permite que seja utilizado o protocolo mais adequado em cada cenário de aplicação, ou mesmo que o Serviço seja realizado totalmente no espaço de endereçamento do Cliente. Essa medida provê maior flexibilidade às aplicações envolvidas.

Essa flexibilidade na utilização de protocolos é importante haja vista que, conforme observado em [Waldo, 2000], o projeto de um protocolo é freqüentemente um compromisso entre generalidade e eficiência. Em sistemas centralizados em torno de um único protocolo, as decisões tendem a favorecer a generalidade. Assim, por mais que um protocolo seja projetado para atender a todos os

cenários atualmente concebíveis, ele ainda é passível de ter um desempenho inferior a um protocolo específico a um daqueles possíveis cenários ou até mesmo ser incapaz de atender a um cenário futuro.

De fato, assim como um dos benefícios do paradigma de programação orientada-a-objetos é a separação clara entre interface e implementação, JINI promove o benefício da separação entre a interface implementada por um Serviço e o protocolo de rede utilizado por seu *proxy* para comunicar-se com ele. Em outras palavras, uma vez que um Cliente JINI descarrega o código do *proxy* para um Serviço em seu próprio espaço de endereçamento, o protocolo de rede utilizado na comunicação deste *proxy* com aquele Serviço é de responsabilidade apenas destes dois últimos componentes. Em última instância é responsabilidade do Serviço, já que seu *proxy* é parte de sua implementação. O protocolo de rede pode ser diferente para diferentes implementações de um mesmo Serviço, ou mesmo mudar ao longo do tempo para um mesmo Serviço.

Mecanismos de geração e manipulação de eventos remotos também são providos pela tecnologia JINI. A partir da definição de um evento remoto, aplicações podem ser projetadas de forma a gerar eventos daquele tipo e/ou serem notificadas acerca dos mesmos.

O LUS, por exemplo, permite às aplicações interessadas serem notificadas acerca de eventos tais como o registro de um novo Serviço que satisfaça certos requisitos. Esses requisitos são expressos por um *template* de consulta, e o registro desse novo Serviço é compreendido como sua adesão à Comunidade capitaneada pelo LUS. Adicionalmente, a remoção de Serviços da Comunidade ou a modificação de um ou mais de seus atributos pode gerar eventos a serem transmitidos às aplicações interessadas.

O mecanismo de manipulação de eventos se dá mediante o registro de *Ouvidores de Eventos* junto às aplicações que os geram. De forma a não desperdiçar recursos registrando-se Ouvidores para aplicações que outrora experimentaram uma interrupção abrupta ou suave em sua execução, o mecanismo de notificação de eventos remotos providos por JINI também faz uso extensivo de *leasing*.

JINI provê também mecanismos para utilização de transações, promovendo as denominadas propriedades ACID (*Atomicity, Consistency, Isolation and Durability*). A implementação de referência provida pela Sun Microsystems para um Gerente de Transações denomina-se `mahalo`.

Originalmente, o modelo de programação definido para JINI estabelecia modelos para *leasing*, geração e manipulação de eventos e transações. Os protocolos de adesão/consulta, juntamente com o LUS, consistiam nos componentes básicos de infra-estrutura. A versão dessa tecnologia lançada em junho de 2003 provê novas funcionalidades nessas duas categorias da arquitetura (modelo de programação e infra-estrutura). Além disso, os Serviços disponíveis foram atualizados de forma a assimilar estas novas funcionalidades.

Uma nova funcionalidade infra-estrutural provida a partir daquela versão é JERI (*JINI Extensible Remote Invocation*), uma nova implementação do modelo de programação RMI. JERI tem a mesma semântica que o JRMP (*Java Remote Method Procotol*), o protocolo *default* para implementação da RMI em Java, mas possui uma estrutura diferente deste último.

O propósito da JERI é expor mais de sua estrutura do que é analogamente realizado pelo JRMP, de forma que os vários níveis que compõem essa estrutura possam ser personalizados. A Arquitetura

RMI é composta por três níveis: o *nível de embaralhamento*, o *nível de invocação de Objetos* e o *nível de transporte*. No nível de embaralhamento, ocorre o embaralhamento de argumentos e de valores de retorno, assim como o disparo de exceções. O nível de invocação identifica o Objeto remoto para o qual deseja-se realizar uma chamada. Por fim, o nível de transporte implementa um protocolo de nível de mensagem para envio da chamada e recepção de sua respectiva resposta.

Dois daqueles níveis podem ser customizados em JERI. No nível de transporte, um número bem maior de opções para transporte dos dados podem ser escolhidas, quando comparado ao JRMP. Opções que não sejam baseadas em *sockets* ou TCP podem ser implementadas, por exemplo. Já o nível de embaralhamento é totalmente exposto, de forma que é possível passar parâmetros implícitos, dados adicionais, codificar os métodos e seus argumentos diferentemente, pôr verificações de controle de acesso, etc.

Alguns problemas podem surgir a partir da adoção da tecnologia JINI no projeto de um sistema distribuído. Felizmente, boa parte deles já possui soluções, quer sejam elas providas pelas versões mais recentes de JINI, quer sejam elas providas por Projetos específicos surgidos no âmbito da Comunidade JINI. Em particular, alguns desses problemas são:

- *sobrecarga nos clientes*: para hospedar um *proxy* para um Serviço JINI, um Cliente deve possuir uma JVM (*Java Virtual Machine*) e um conjunto de APIs. Muito embora existam diferentes opções de implementação de JVM e de conjuntos de APIs suportadas, o montante de recursos computacionais consumidos por estes componentes pode inviabilizar a utilização da tecnologia JINI. Este problema pode ser contornado com a adoção do Projeto *Surrogate*, a ser comentado no próximo capítulo;

- *maior vulnerabilidade a ataques do tipo DoS (Denial of Service)*: embora a plataforma Java possua um modelo de segurança extensível que previne código não-confiável de realizar certas ações que podem representar ataques de segurança, esse modelo não previne certos ataques de DoS. Um *proxy* para um Serviço poderia, por exemplo, inicializar *threads*, alocar memória, ou abrir *sockets* até que recursos do Cliente sejam completamente consumidos, o que poderia desqualificar esta abordagem. Este problema pode ser evitado com a utilização do modelo de Segurança provido pelas versões mais recentes de JINI, que será abordado na próxima seção deste capítulo;

- *problemas de escalabilidade nos Servidores*: para se ter certeza de que um Servidor está apto a atender concorrentemente um grande número de solicitações de Clientes, o desenvolvedor do Serviço precisa estar apto a controlar como e quando recursos finitos (tais como *threads*, *sockets* e manipuladores de arquivos) são alocados para o atendimento daquelas solicitações. Uma das formas de tratar esse problema é a disponibilização de vários Serviços, de forma a efetuar uma distribuição da carga de trabalho entre eles;

- *tempo de descarregamento de proxies para Serviços*: esse tempo é relacionado à taxa de transmissão disponível e ao volume de dados transmitidos, independente da natureza destes dados. Os dados podem consistir de PDUs SNMP, páginas HTML (*HyperText Markup Language*) ou *bytecodes* alusivos a *proxies* para Serviços. Em particular, o efeito do tempo de

descarregamento destes *proxies* pode ser amenizado segundo algumas estratégias utilizadas na transmissão de documentos Web, tais como:

- manter o código do proxy tão pequeno quanto possível;
- sempre que possível, procurar dividir as funcionalidades oferecidas de forma que partes individuais possam ser descarregadas apenas sob demanda;
- utilizar estratégias de *caching* bem projetadas, de forma a não descaracterizar a aplicação da tecnologia JINI.

3.4 Aspectos básicos de segurança

Em sistemas Cliente/Servidor tradicionais, o código executado pelo Cliente é inteiramente confiável. Não há necessidade, portanto, de efetuar decisões de autorização com relação e este código.

A utilização da tecnologia JINI para a construção desses sistemas, porém, faz com que algumas questões de segurança outrora ausentes venham à tona. No âmbito da Comunidade JINI, o Projeto *Davis* [Scheifler e Blackman, 2005] foi criado para endereçar estas questões e propor soluções para elas. As melhorias de segurança introduzidas a partir deste Projeto foram aprovadas pela Comunidade e incorporadas na versão do JTSK (*JINI Technology Starter Kit*) lançada em junho de 2003.

Segundo o modelo de segurança surgido a partir do Projeto Davis, um *proxy* descarregado a partir de uma fonte não-confiável deve ser *preparado* antes de qualquer operação remota possa ser invocada através dele. Esta preparação é composta pelos seguintes passos:

1. verificação de confiança;
2. fornecimento dinâmico de permissões;
3. imposição de restrições.

O primeiro passo no processo de preparação de um *proxy* é a verificação de que ele é confiável. Esse passo é importante haja vista que as operações realizadas através deste *proxy* podem produzir efeitos significativos no mundo real. Caso o *proxy* não seja confiável, não há garantias de que essas operações estejam sendo corretamente realizadas, ou mesmo se elas estão sequer sendo realizadas de fato.

Um exemplo disso é que, numa operação remota de transferência monetária, caso não seja verificada a confiança num pretenso *proxy* para o Serviço que a implementa, não há garantias de que um mecanismo prévio de autenticação possa estar sendo silenciosamente ignorado e que a operação possa estar sendo maliciosamente manipulada.

Quando o código de um *proxy* é descarregado a partir de uma fonte potencialmente não-confiável, o mecanismo utilizado para verificar se ele é confiável baseia-se na premissa de que somente o próprio Serviço é capaz de verificar a confiança em um *proxy* para ele.

Esse mecanismo de verificação de segurança, ilustrado pela Figura 3.2, inicia-se com a obtenção de um *proxy de bootstrap* mediante comunicação com o *proxy* para o qual deseja-se verificar confiança (a). Em seguida, o *proxy* de *bootstrap* é investigado por um *Objeto Verificador de Confiança* local para ter-se a certeza de que ele próprio é confiável (b). Nessa investigação, pode ser utilizado um algoritmo baseado na checagem recursiva de todos os seus campos, de forma a identificar se todos os tipos encontrados são locais. Essa investigação é amparada na premissa de que todo código local é confiável.

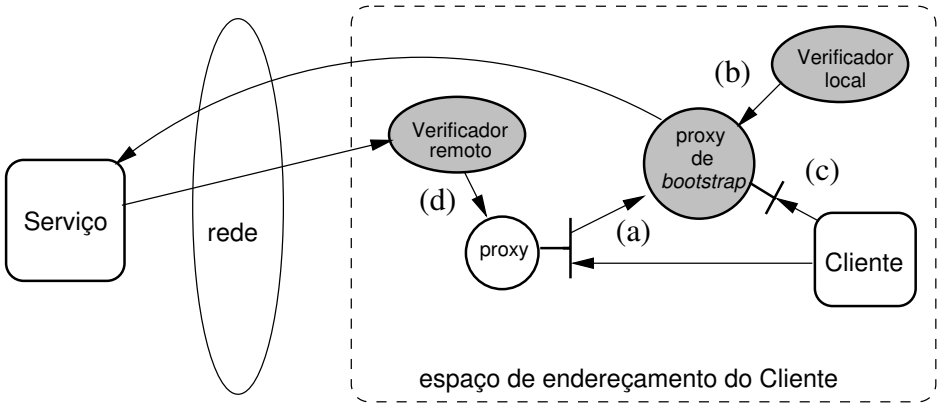

Figura 3.2: Verificação de confiança em um *proxy* descarregado a partir de uma fonte não-confiável.

Após ser verificada a confiança no *proxy* de *bootstrap*, procede-se à invocação remota de um método neste *proxy* junto ao Serviço, que retorna um outro Objeto Verificador de Confiança (c). Por fim, este Objeto avalia se o *proxy* originalmente descarregado é ou não de fato confiável (d). Obviamente, deve-se ter certeza de que o código desse Verificador de Confiança remoto não foi corrompido em seu trajeto até o espaço de endereçamento do Cliente. Esse processo, conhecido como a garantia da integridade do código descarregado, será abordado posteriormente.

Uma alternativa à obtenção de um Objeto Verificador de Confiança remoto seria a transmissão do *proxy* sob investigação para o espaço de endereçamento do Serviço, de forma que a verificação seria feita lá, e não no espaço de endereçamento do Cliente. O problema com essa abordagem é que existe a possibilidade de um *proxy* malicioso serializar-se de tal forma que ele seria perfeitamente tido como confiável pelo Serviço. Assim, de forma a detectar mais precisamente a verdadeira natureza de um *proxy*, achou-se mais prudente enviar um Objeto Verificador de Confiança para o espaço de endereçamento do Cliente e examinar este *proxy* na forma segundo a qual ele será utilizado pelo Cliente.

O próximo passo a ser seguido na preparação de um *proxy* é a garantia de permissões a ele. Em JINI, as permissões de acesso aos recursos computacionais do Cliente são dadas ao *proxy* dinamicamente, e uma das razões pelas quais isso ocorre é que não se sabe, a princípio, a partir de onde o código relativo ao *proxy* será descarregado. Esse processo de autorização deve ser realizado apenas após a verificação de confiança no *proxy*, de forma a evitar que *proxies* mal-intencionados utilizem aquelas permissões para causar danos aos Clientes.

No entanto, o modelo de segurança Java não possui a noção de garantia de permissão a códigos, mas sim ao *Domínio de Proteção* ao qual o código pertence. Na versão 1.4 da J2SE (*Java 2 Platform Standard Edition*), a noção de um Domínio de Proteção foi estendida tendo em vista o modelo de segurança de JINI. Um Domínio de Proteção agora tem três componentes: *principals*, *code source* e *class loader*.

O *principal* identifica quem está executando o código. O *code source* é a noção tradicional de onde o código veio, uma URL (*Uniform Resource Locator*), e de quem assinou o código. O *class loader* é o Objeto que carregou o código. Assim, as permissões são dadas não ao código do *proxy*, mas ao *class loader* que carrega este código.

Uma vez que confiança foi verificada e permissões foram dadas ao *proxy*, *restrições* devem ser impostas para garantir o nível desejado de segurança em invocações futuras através deste *proxy*.

De fato, uma implementação de um determinado Serviço JINI pode impor restrições de segurança acerca de como (ou quais) Clientes interagem com ela. Ela pode requerer, por exemplo, que um Cliente autentique-se antes de qualquer interação. Serviços podem até mesmo recusar-se a interagir com Clientes que eles não reconhecem ou não desejam. Similarmente, Clientes também podem requerer autenticação do Serviço antes de invocar seus métodos, bem como podem insistir na garantia da integridade dos dados trocados. Essas restrições podem ser impostas no *proxy* tanto pelo Serviço (antes de exportar o *proxy*) quanto pelo Cliente (após descarregar esse *proxy* da rede).

As restrições são aplicadas ao *proxy* na granularidade de seus métodos. Um determinado método pode possuir um conjunto de restrições, as quais podem ser divididas em restrições *obrigatórias* e *preferenciais*, que dão uma idéia sobre o grau de impacto do seu atendimento para a invocação do método.

A versão 2.0 do JTSK define um conjunto padrão de restrições abrangendo integridade das mensagens trocadas, autenticação de Clientes e Serviços e delegação (quando Serviços invocam terceiros em nome de Clientes). Outro conjunto padrão de restrições define as identidades com as quais um Cliente ou Serviço precisa autenticar-se. E algumas poucas restrições relacionam-se não com aspectos de segurança, mas sim com aspectos de QoS.

A definição de novos tipos personalizados de restrições é possível. No entanto, recomenda-se cautela sobre essa medida, sob pena destas restrições não serem compreendidas por todas as partes envolvidas e assim inviabilizar a comunicação entre elas. Outra medida que contribui para a inoperância da comunicação é a definição de restrições conflitantes. Por exemplo, um Serviço pode requerer obrigatoriamente que o Cliente autentique-se, enquanto o Cliente faz uso de uma restrição obrigatória sobre o seu anonimato.

Independentemente da natureza dessas restrições, porém, elas são expressas de forma abstrata. Isso faz com que as implementações de segurança sejam de fato plugáveis, suportando uma variedade de protocolos, tais como o SSL (*Secure Socket Layer*) [Dierks e Allen, 1999] ou Kerberos [Kohl e Neuman, 1993].

Outro aspecto coberto pelo modelo de segurança proposto para JINI é a integridade do código. No modelo de programação RMI, quando um Objeto é trocado entre aplicações, seus dados são enviados como parte de uma mensagem. Os arquivos de Classe que compõem seu código são descarregados a partir de um Servidor de *codebase* (usualmente um Servidor HTTP) com base num

URL contida na mensagem.

A abordagem alternativa seria a transmissão do código dos Objetos juntamente com seus dados. Porém, isso seria proibitivo por duas razões. A primeira delas é que, tomando como base uma Comunidade JINI, a obtenção de todo e qualquer código seria intermediada pelo seu respectivo LUS, o que talvez comprometesse a escalabilidade daquela Comunidade.

A outra razão é que na arquitetura de segurança da Plataforma Java 2 pode-se associar garantias de permissão ao código descarregado a partir daquela URL. Se o código de um Objeto é transmitido em uma mensagem (juntamente com os seus dados) e não há nenhuma URL associada a ele, não há noção sobre a sua natureza, sob uma perspectiva de segurança.

O problema com a abordagem baseada em URLs HTTP é que não há nada intrínseco àquele protocolo que garanta segurança. De fato, não há virtualmente nada que impeça a interceptação do código de um Objeto por uma entidade mal-intencionada e a substituição daquele código por outro potencialmente danoso ao Cliente que usará o Objeto. Além disto, essa mesma entidade poderia personificar um Servidor HTTP legítimo e enviar aquele código maléfico aos Clientes que entrassem em contato com ela pensando estarem comunicando-se com o Servidor HTTP original.

Uma abordagem possível para conseguir-se integridade de código seria utilizar URLs HTTPS. O HTTPS (*HTTP over SSL*) [Rescorla, 2000] é um protocolo desenvolvido pela Netscape Communications Corp. e incorporado em seu navegador Web para encriptação/decriptação de páginas trocadas entre Clientes e Servidores Web. O problema apontado na utilização do HTTPS é a necessidade de uma infra-estrutura capaz de prover chaves públicas de forma a autenticar o Servidor, o que inviabilizaria a utilização de Clientes simples. Além disso, com HTTPS todo o código será sempre encriptado, o que nem sempre é um requisito necessário.

Uma abordagem alternativa ao uso de HTTPS, criada a partir do Projeto Davis, é a utilização de URLs HTTPMD (*HTTP Message Digest*). Um exemplo de uma URL HTTPMD é ilustrado abaixo:

```
httpmd://myserver/verifier_code.jar;md5=9bf342fda42354379098099801f1f234234
```

Conforme observado, uma URL HTTPMD contém um *checksum* criptográfico do código do Objeto, que é encapsulado num arquivo JAR (*Java ARchive*). Este *checksum* é calculado a partir de um algoritmo conhecido. No exemplo dado, o *checksum* foi obtido a partir da aplicação do algoritmo MD5.

A Figura 3.3 ilustra como é feita a aquisição do código e a verificação de sua integridade, tomando como exemplo o descarregamento dinâmico do código de um Objeto Verificador de Confiança realizado por um *proxy* de *bootstrap*.

Conforme observado, ao descarregar o código a partir de um Servidor HTTPMD (a), o *proxy* obtém o seu *checksum* criptográfico aplicando nele o algoritmo MD5 (b). Em seguida, ele compara o *checksum* obtido com aquele presente na URL HTTPMD (c). Caso verifique-se igualdade entre ambos, o código descarregado pode ser considerado íntegro. Caso contrário, existe a possibilidade do código e/ou do *checksum* terem sido corrompidos ou mesmo substituídos no seu caminho até o espaço de endereçamento do Cliente, o que inviabilizaria o prosseguimento das atividades.

A implementação alternativa do modelo de RMI citada na seção anterior, JERI, suporta todas as funcionalidades providas pelo modelo de segurança surgido a partir da versão 2.0 da tecnologia

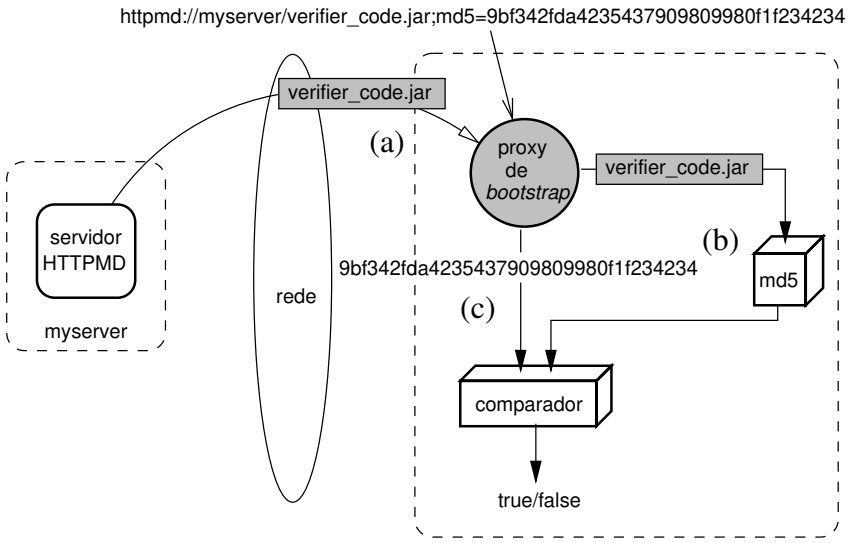

Figura 3.3: Verificação da integridade do código descarregado a partir de uma URL HTTPMD.

JINI. Em particular, quase todas as restrições que podem ser impostas (autenticação, confidencialidade e integridade nas comunicações) são mecanismos de segurança do nível de transporte. A única funcionalidade que é implementada num nível superior é a integridade de código, implementada no nível de embaralhamento.

Um dos mais importantes aspectos globais do modelo de segurança provido pelas versões mais recentes de JINI é que decisões de segurança, que refletem-se em última instância no uso das funcionalidades descritas ao longo desta seção, são feitas no tempo de emprego das aplicações sem que haja nenhuma alteração em seus respectivos códigos. Esse benefício é ainda mais notável quando vislumbra-se uma plêiade de cenários de emprego para tais aplicações, cada um destes cenários possuindo requisitos de segurança específicos.

3.5 Conclusão

Este capítulo descreveu os conceitos básicos por trás de JINI, uma tecnologia proposta pela Sun Microsystems para a construção de sistemas distribuídos auto-resilientes. Após um breve histórico, foram descritos os aspectos de arquitetura, comunicação e segurança que essa tecnologia possui. Tal descrição baseou-se não apenas nas especificações referentes à JINI, mas também em considerações feitas por renomados desenvolvedores de software, em particular Bill Venners[1], Frank Sommers[2] e Jim Waldo[3]. Essas considerações são disponibilizadas gratuitamente a partir de

[1] Bill Venners é o proprietário do Projeto *ServiceUI*, a ser descrito no próximo Capítulo.

[2] Frank Sommers participou de vários Projetos relacionados ao de JINI na área de computação baseada em *clusters*.

[3] Jim Waldo, conforme citado no início deste Capítulo, foi um dos nomes por trás do surgimento de JINI.

`http://www.artima.com`, o *site* da empresa Artima, Inc., de propriedade de Bill Venners.

No que diz respeito aos aspectos arquiteturais e de comunicação de JINI, foi visto que Serviços registram seus respectivos *proxies* junto a um ou mais LUSs, de forma a serem acessados posteriormente por Clientes. Foi visto também que a manutenção deste registro se dá através da renovação periódica de um *lease* e que, uma vez que um Cliente adquire um *proxy* para um Serviço, a comunicação com esse Serviço (se ocorrer) pode ser realizada utilizando-se virtualmente qualquer protocolo. Mecanismos para geração/manipulação de eventos e suporte ao uso de transações também foram abordados.

Com relação ao modelo de segurança contido nas versões mais recentes de JINI, foi visto que, antes da realização de qualquer invocação junto a um *proxy*, um conjunto ordenado de procedimentos deve ser realizado. Esses procedimentos consistem na verificação da confiança no *proxy*, na garantia de permissões a ele e no estabelecimento de restrições para a realização das invocações.

Como um exemplo de viabilidade deste modelo, uma implementação alternativa da semântica RMI, denominada JERI, foi disponibilizada junto ao JTSK. O modelo de segurança definido atualmente para JINI permite o emprego imediato de mecanismos tais como SSL e Kerberos para garantir autenticação, integridade e confidencialidade nas interações. Além disso, URLs HTTPMD podem ser utilizadas para garantir a integridade do código descarregado dinamicamente a partir da rede.

No próximo capítulo será descrito de que formas essa poderosa tecnologia pode ser empregada especificamente na construção de NMSs.

Capítulo 4

Considerações gerais sobre a utilização de JINI na construção de NMSs

4.1 Introdução

O Capítulo 2 abordou algumas das evoluções realizadas no âmbito da IETF na área de gerência de redes, enquanto o Capítulo 3 introduziu os conceitos básicos por trás da tecnologia JINI.

Este capítulo descreve as considerações gerais que devem ser levadas em conta quando JINI é empregada na construção de NMSs. Em particular, essas considerações foram aplicadas na construção de duas provas de conceitos, descritas em capítulos subseqüentes.

4.2 Associação de Serviços JINI a sistemas gerenciados

Quando é considerada a utilização de JINI na construção de NMSs, a idéia imediata é a utilização de Serviços JINI associados a sistemas gerenciados. De fato, diferentes níveis de granularidade podem ser usados no projeto de Serviços JINI para sistemas gerenciados, conforme é ilustrado pela Figura 4.1.

Conforme ilustrado pela Figura 4.1, podem ser associados a um sistema gerenciado tantos Serviços JINI quanto forem desejados.

O maior nível de granularidade é aquele no qual cada Serviço JINI é equivalente a um MO. Assim, a Figura 4.1 mostra a direita um dispositivo gerenciado pelos Serviços JINI $S_1, S_2, ..., S_n$, que são equivalentes, em termos de modelagem, aos vários MOs que seriam definidos para a gerência daquele dispositivo utilizando técnicas tradicionais.

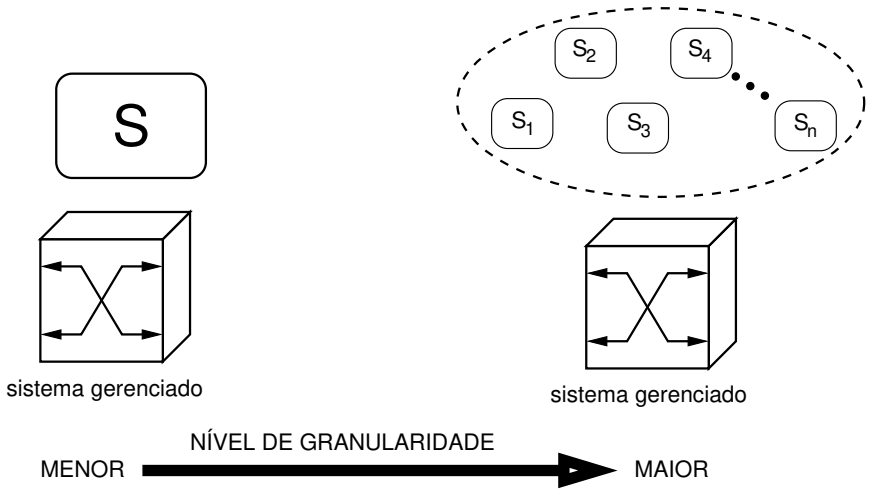

Figura 4.1: Variação do nível de granularidade no projeto de Serviços JINI.

Modelar Serviços JINI na granularidade de MOs pode trazer uma manutenção mais fluida nas operações de um NMS. Isso porque cada um desses Serviços é independente de qualquer outro, e as considerações de projeto são feitas basicamente no nível da interface a ser implementada e dos atributos que o Serviço possuirá.

Porém, um cuidado especial deve ser tomado quando se adota esse tipo de modelagem, porque ela pode levar a um número elevado de Serviços JINI associados a um único sistema gerenciado. Já que cada um destes Serviços é executado por uma instância diferente da JVM, o melhor seria construí-los de forma a utilizar ao máximo o recurso da ativação.

Independentemente, deve-se levar em conta que a operação simultânea de todos aqueles Serviços consumiria mais recursos computacionais do que um único Serviço JINI equivalente em termos de funcionalidade. Além disso, quanto maior for o número de Serviços associados a um determinado sistema, maior será o tráfego de rede associado a sua gerência.

O menor nível de granularidade é aquele no qual o sistema é gerenciado com o auxílio de um único Serviço JINI. Esse nível é ilustrado pelo extremo esquerdo da Figura 4.1.

Quando se gerencia um sistema através de um único Serviço JINI, perde-se um pouco em termos de fluidez na manutenção, pois alterações funcionais no Serviço irão repercutir diretamente no projeto de sua interface e/ou no valor de seus atributos.

Além disso, pode-se carregar para a memória muito mais do que é normalmente necessário, pois pode haver aspectos de gerência em um sistema que, embora previstos e suportados, são raramente utilizados.

Porém, a utilização de um único Serviço JINI para a gerência de um sistema pode mostrar-se vantajosa frente à fragmentação desse Serviço em vários outros transientes quando esses vários outros Serviços são freqüentemente utilizados. Isso ocorre porque, conforme comentado anteriormente, cada um desses Serviços é executado por uma instância diferente da JVM e o consumo acumulativo de recursos computacionais nesse caso seria bem maior do que aquele verificado por

um único Serviço JINI equivalente.

Além disso, a depender do projeto da interface implementada pelo Serviço, o tráfego de rede gerado a partir dele pode ser inferior àquele verificado para a gerência dos vários Serviços que seriam equivalentes a ele utilizando um nível de granularidade maior.

4.3 Agrupamento de Serviços em Domínios Gerenciados

Conforme visto no Capítulo 3, Serviços JINI são agrupados em Comunidades. Seguindo a proposição em monitorar e controlar sistemas gerenciados com o auxílio de Serviços JINI, nada mais natural do que agrupá-los em Comunidades.

A Figura 4.2 mostra uma infra-estrutura de rede na qual existe um Serviço JINI associado a cada sistema gerenciado, sendo empregado portanto o menor nível de granularidade na modelagem daqueles Serviços.

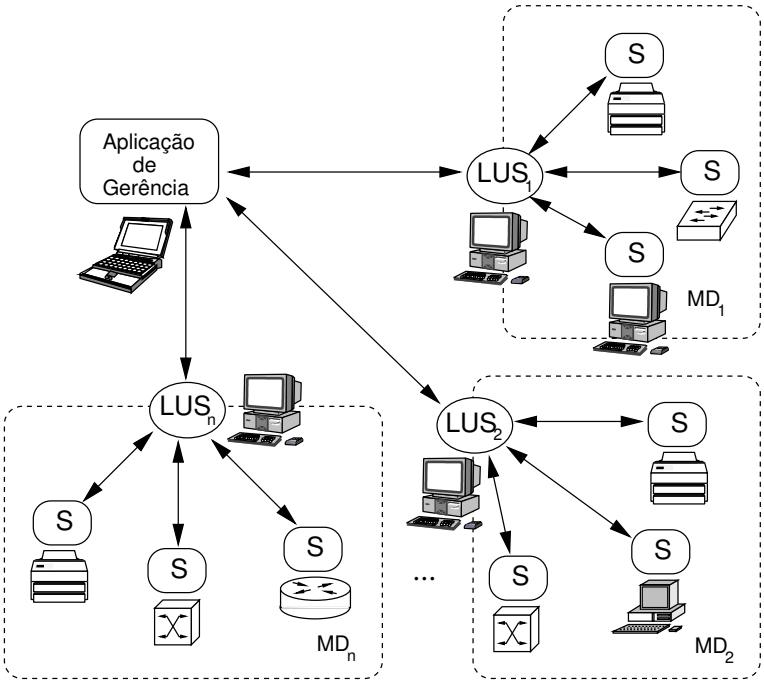

Figura 4.2: Serviços JINI agrupados em Domínios Gerenciados federados.

Cada um dos Serviços pertence a uma Comunidade, e, para os propósitos de gerência, cada Comunidade é denominada de Domínio Gerenciado, ou MD (*Managed Domain*). A Figura 4.2 ilustra os Domínios Gerenciados $MD_1, MD_2, ..., MD_n$, cada um deles sob a responsabilidade de um LUS. Serviços JINI são neles agrupados com base em princípios administrativos e geográficos.

A Figura 4.3 ilustra as interações verificadas num Domínio Gerenciado. Cada um dos Serviços JINI associados a um sistema gerenciado inicialmente registra seu *proxy* junto ao LUS, fazendo parte então do Domínio Gerenciado (a). Conforme visto no Capítulo 3, a manutenção desse registro se dá através da renovação periódica de um *lease*.

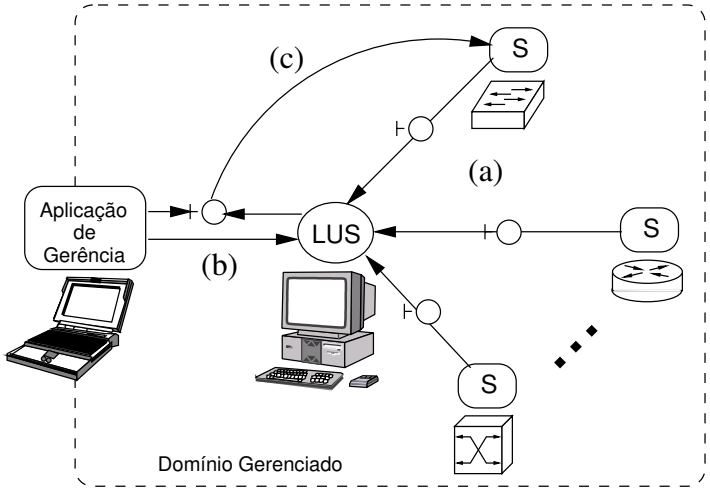

Figura 4.3: Detalhe das interações verificadas num Domínio Gerenciado.

A Aplicação de Gerência comunica-se com o LUS para a obtenção de um *proxy* para um dos Serviços pertencentes àquele Domínio (b). Essa busca tem normalmente como parâmetros a interface implementada pelo Serviço e alguns de seus atributos, tais como a localização ou nome administrativo do sistema gerenciado ao qual está associado o Serviço JINI, por exemplo.

Após ter obtido um ou mais *proxies* junto ao LUS, a Aplicação de Gerência pode invocar operações remotas junto a esses *proxies*, efetuando a monitoração e controle do sistema gerenciado (c).

Apesar da Figura 4.3 ilustrar uma comunicação comum entre a Aplicação de Gerência e Serviços JINI associados aos sistemas gerenciados, alguns pontos merecem destaque. O primeiro deles é que, conforme citado no Capítulo 3, virtualmente qualquer protocolo pode ser utilizado na comunicação entre aquelas aplicações. Poderia ser utilizado inclusive qualquer uma das versões do SNMP, por exemplo.

O segundo ponto notável diz repeito às formas de comunicação entre a Aplicação de Gerência e os Serviços JINI destinados à gerência dos sistemas. De fato, o registro de Ouvidores de Eventos junto aos Serviços permite que eventos remotos sejam transmitidos às Aplicações de Gerência. Isso permite que, não só a estratégia de *polling* mas também a estratégia de notificações, possam ser implementadas em NMSs baseados na tecnologia JINI.

Por fim, a associação dos mecanismos de *leasing* com os de eventos remotos permite garantir a auto-resiliência desses sistemas como um todo. Em particular, caso a Aplicação de Gerência registre Ouvidores de Eventos específicos junto a um LUS responsável por um determinado Domínio

Gerenciado, essa Aplicação vai estar ciente da inserção ou remoção de Serviços nesse Domínio Gerenciado, tendo assim uma visão atualizada do cenário de atuação.

4.4 Incorporação de Interfaces com o Usuário nos Serviços

No projeto tradicional de um software de gerência, uma MIB é considerada o resultado do pensamento de especialistas acerca de quais são os pontos mais notáveis para a monitoração e o controle de um sistema ou de determinados aspectos deste sistema, conforme comentado em [Wellens e Auerbach, 1996].

Esses pontos muitas vezes endereçam características específicas do fabricante, normalmente aquelas características que tornam o sistema diferente de sistemas similares produzidos por outros fabricantes.

O problema com isso é que na maioria das vezes Aplicações de Gerência de propósito geral não tomam conhecimento desses pontos e não podem endereçá-los através de PDUs SNMP.

Tem-se então um cenário indesejável no qual a gerência daqueles sistemas se faz localmente, ou remotamente através do protocolo TELNET [Postel e Reynolds, 1983]. Muitas vezes, uma solução paliativa é dada por meio de navegadores de MIB, mas sem nenhum resultado expressivo.

Em [Bruins, 1996] é sugerida uma abordagem baseada na utilização de servidores HTTP embutidos em sistemas gerenciados, que exportavam páginas codificadas em HTML (*Hyper Text Markup Language*) contendo informações de gerência para navegadores Web.

A principal vantagem provida por aquela abordagem é que os sistemas gerenciados podiam exportar GUIs (*Graphical User Interfaces*) semanticamente ricas para a sua monitoração e controle, dispensando o uso de ferramentas de gerência personalizadas.

Apesar da enorme vantagem oferecida por aquela abordagem, deve se ter sempre em mente que ela também baseia-se na definição estrita de um protocolo e um modelo de dados. Naquele caso, as informações de gerência são transferidas utilizando-se HTTP e codificadas em HTML.

Possuindo inspiração na idéia exposta em [Bruins, 1996], é sugerida a incorporação do Projeto *ServiceUI* [Venners, 2005], surgido no âmbito da Comunidade JINI, para tornar possível a importação dinâmica de Interfaces com o Usuário semanticamente ricas a partir dos Serviços JINI associados aos sistemas gerenciados.

A idéia por trás do Projeto *ServiceUI* é a de embutir Objetos de Interface com o Usuário nos *proxies* para os Serviços JINI, mais precisamente nos seus atributos. Nesse contexto, tais Objetos são também chamados de *Objetos Adaptadores de Usuário*, porque eles adaptam a interface provida por um *proxy* a alguma interface compreensível pelos usuários. A incorporação dessa idéia na modelagem de Serviços JINI associados aos sistemas gerenciados é ilustrada pela Figura 4.4.

Através da incorporação do Projeto *ServiceUI*, consegue-se o mesmo efeito observado na proposta descrita em [Bruins, 1996], porém com duas vantagens adicionais frente àquela iniciativa.

A primeira dessas vantagens remete ao próprio uso da tecnologia JINI, mais precisamente ao fato de que não há nenhuma restrição ao protocolo utilizado na comunicação entre *proxies* e seus Serviços associados.

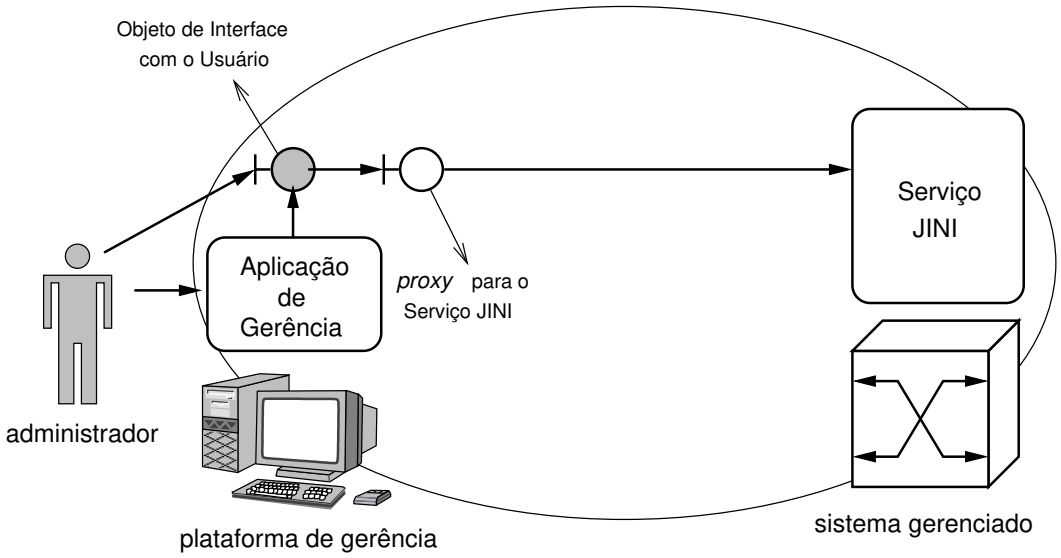

Figura 4.4: Incorporação do Projeto *ServiceUI*.

Assim, contrariamente à abordagem vista em [Bruins, 1996], é possível que qualquer protocolo, não apenas HTTP, seja utilizado para transmissão dos dados de gerência. Inclusive, para cenários com restrições de segurança, a versão simplificada do HTTP não pode ser utilizada.

A segunda vantagem remete ao próprio Projeto *ServiceUI*, que não impõe qualquer restrição ao tipo da Interface com o Usuário provida pelo Objeto correspondente. Assim, a incorporação deste Projeto em NMSs baseados em JINI permite aos usuários daqueles sistemas utilizar uma interface gráfica, uma interface de texto, uma combinação de texto e gráficos ou até mesmo um ambiente de Realidade Virtual. De fato, o Objeto é dito ser simplesmente de Interface com o Usuário, e não de Interface *Gráfica* com o Usuário.

Apesar dos comentários feitos nesta subseção serem limitados à comparação com a abordagem exposta em [Bruins, 1996], há outras considerações importantes sobre a utilização de Objetos de Interface com o Usuário.

Seguem algumas desvantagens da abordagem baseada em *proxies* e Objetos de Interface com o Usuário, em detrimento da comunicação Cliente/Servidor tradicional envolvendo páginas HTML transferidas usando o HTTP:

- *Serviços JINI não são tão fáceis de serem construídos quanto páginas HTML*: enquanto uma simples página Web pode ser construída a partir do conhecimento de algumas poucas *tags*, Serviços JINI podem ser construídos somente a partir de um conhecimento prévio de programação Orientada a Objetos, fundamentos da linguagem de programação Java e conhecimento básico da API disponibilizada no JTSK. Muito embora o trabalho de desenvolvimento destes Serviços seja amenizado por ferramentas do tipo IDE (*Integrated Development Environment*), aqueles requisitos não mudarão;

- *tempo de descarregamento*: o *overhead* causado pelo tempo de descarregamento do código referente ao *proxy* para um Serviço é um obstáculo relevante ao seu emprego. Isso é particularmente verdade caso considere-se que um Cliente humano queira utilizar aquele Serviço. Curiosamente, as maneiras de endereçar-se este problema são análogas àquelas adotadas para o projeto de páginas Web: manter sob controle o tamanho do código do *proxy*, dividir sempre que possível as funcionalidades providas e utilizar estratégias de *caching*.

Por outro lado, as vantagens de embutir-se Objetos de Interface com o Usuário em *proxies*, separando a forma de apresentação da funcionalidade oferecida por seus respectivos Serviços, podem ser sumarizadas com se segue:

- *facilidade de automação não prejudica a utilização direta dos proxies*: um mesmo *proxy* pode ser igualmente acessível por quaisquer Clientes, sejam eles humanos ou não;

- *facilidade de evolução*: visto que as Interfaces com o Usuário sofrem modificações mais freqüentemente do que a interface para o Serviço, a separação entre ambas garante que mudanças nas primeiras não afetem o projeto da última;

- *extensão do alcance do Serviço*: a possibilidade de associar-se diferentes interfaces a um mesmo *proxy* permite que o seu Serviço correspondente seja disponibilizado em equipamentos tão distintos como uma TV, um computador convencional ou um telefone celular, por exemplo;

- *formas de interação mais naturais com o usuário*: quando Serviços são acessados através de um Objeto de Interface com o Usuário, essa interface pode potencialmente assumir qualquer forma, inclusive aquela mais natural para o usuário. Serviços de *Email* e *Chat* poderiam incluir uma interface de voz, por exemplo;

- *utilização mais suave por parte do usuário*: visto que *proxies* encapsulam o comportamento necessário à execução dos seus respectivos Serviços, os usuários não confrontam-se com problemas tais como aqueles observados quando um componente de software (uma imagem TIFF, por exemplo) não pode ser manipulado por um navegador Web e um *plug-in* precisa ser descarregado pelo Cliente.

4.5 Provisão de interações seguras entre Aplicações de Gerência e Serviços

Conforme abordado no Capítulo 3, JINI possui atualmente um modelo de segurança que possibilita a imposição de restrições em *proxies* e é capaz de garantir a integridade do código descarregado.

Há dois parâmetros a serem considerados na verificação da necessidade da utilização daqueles mecanismos de segurança em Serviços JINI associados a sistemas gerenciados.

O primeiro daqueles parâmetros é o nível de segurança do ambiente no qual as interações entre a Aplicação de Gerência e os Serviços JINI ocorrem. Ambientes seguros não demandam a utilização de mecanismos de segurança entre Aplicações de Gerência e Serviços JINI. Por outro lado, ambientes inseguros demandam a utilização daqueles mecanismos para a garantia de comunicações seguras.

Quando ambientes inseguros são considerados, e portanto há uma demanda pela utilização de mecanismos de segurança, o segundo parâmetro é considerado para verificação da aplicação dos mecanismos de segurança providos por JINI. Em particular, é investigado se o protocolo a ser utilizado na comunicação entre Aplicações de Gerência e Serviços JINI já não supre os requisitos de segurança impostos pelo ambiente. Caso ele não os supra, a utilização dos mecanismos de segurança providos por JINI é considerada mandatória.

4.6 Utilização dos mecanismos transacionais na comunicação entre Aplicações de Gerência e Serviços

A utilização de mecanismos transacionais é de suma importância em sistemas distribuídos, haja vista que tais mecanismos provêem, no escopo das propriedades ACID, a consistência sobre um conjunto de operações em um ou mais *Participantes* remotos.

Essa propriedade é importante na gerência de redes, mormente na gerência de configuração. Nessa área funcional, um volume razoavelmente grande de informações é transmitido a dispositivos e Serviços, visando a sua configuração. Assim, de forma a garantir-se configurações consistentes, recomenda-se a aplicação de mecanismos transacionais na transferência daquelas informações.

Conquanto o modelo tradicional possua deficiências nesta área, algo que será abordado posteriormente, a proposta definida nesta Tese a endereça com a aplicação dos mecanismos transacionais providos por JINI.

Tradicionalmente, mecanismos transacionais *garantem* que todos os Participantes implementam corretamente a semântica de uma transação. Na abordagem utilizada por JINI, no entanto, os Participantes implementam essa semântica da forma que lhes convêm. Com efeito, o foco desta abordagem está na coordenação entre esses Participantes. Para tal, ela define um protocolo do tipo 2PC (*Two-Phase Commit*) que *permite* às aplicações implementarem transações.

A Figura 4.5 ilustra a aplicação do protocolo 2PC desenvolvido em um NMS baseado em JINI. Conforme pode-se observar, esse protocolo concebe a utilização de um Gerente de Transações. Inicialmente, um *Cliente* (Aplicação de Gerência) obtém uma referência para este Serviço junto ao LUS. Ele então solicita inicialmente ao Gerente a criação de uma transação (a), e recebe como resposta um Objeto contendo um identificador da transação e um *lease* renovável, cuja expiração fará com que o Gerente de Transações aborte a transação.

Ao efetuar operações em um ou mais Serviços (b), o Cliente envia como parâmetros dessas operações o identificador da transação (ou o Objeto que o contém). Os Serviços comunicam-se com

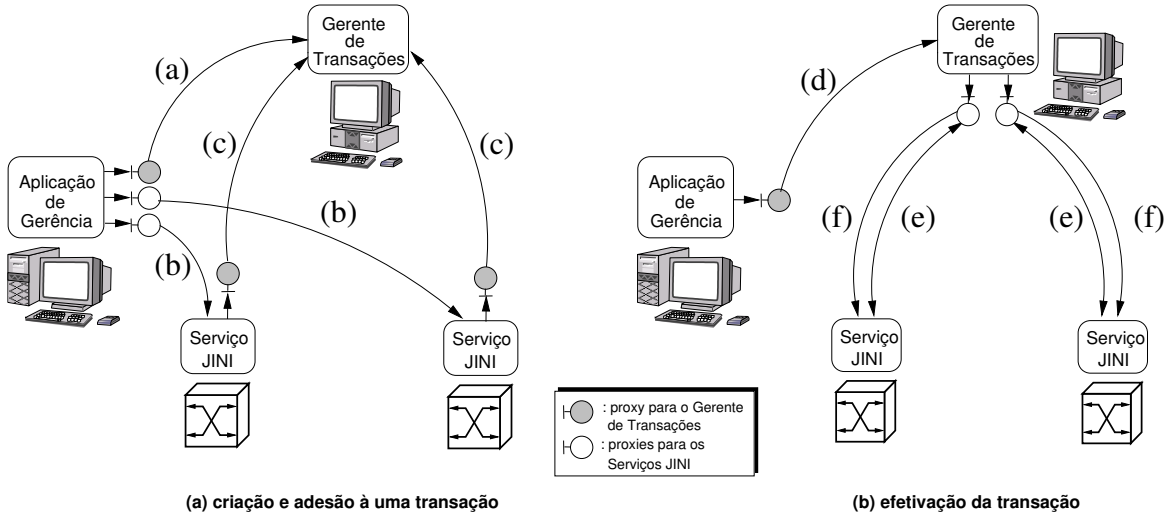

Figura 4.5: Utilização do protocolo 2PC em um NMS baseado em JINI.

o Gerente de Transações e *aderem* à transação como Participantes (c).

A efetivação de uma transação é tipicamente iniciada pelo Cliente, que a solicita junto ao Gerente de Transações (d). Essa efetivação compreende inicialmente um processo de votação envolvendo todos os seus Participantes (e). Supondo-se que todos eles votam em *prepared*, o Gerente de Transações solicita que as alterações solicitadas sejam de fato efetivadas (e). Caso contrário (isto é, se algum deles votar em *abort*), essas alterações serão descartadas.

O suporte à utilização de transações é uma característica importante apresentada pela tecnologia JINI. Os mecanismos transacionais descritos nesta seção podem ser empregados de várias formas no projeto de NMSs baseados nessa tecnologia. Essas formas incluem desde operações envolvendo vários Serviços (como é o caso ilustrado pela Figura 4.5), até operações relacionadas a apenas um Serviço (particularmente quando deseja-se que a consistência seja estabelecida quando um conjunto ordenado de operações são realizadas junto a um Serviço).

4.7 Emprego de dispositivos com baixo poder computacional

Para que um dispositivo (denominação dada a um componente de hardware ou software) utilize os recursos oferecidos pela tecnologia JINI, ele precisa satisfazer alguns requisitos críticos: ele precisa estar apto a participar dos protocolos de descoberta e adesão e precisa estar apto a descarregar e executar Classes Java. Adicionalmente, ele pode ter a necessidade de exportar Classes Java de forma que essas últimas estejam disponíveis para descarregamento por uma entidade remota.

Para muitos dispositivos, os requisitos expostos acima não são difíceis de atender. Porém, há dispositivos que, por uma razão ou outra, não conseguem satisfazer esses requisitos. Esses dispositivos não possuem, portanto, capacidade nativa de comunicar-se ou fazer parte de uma Comunidade

JINI. A fim de prover essa capacidade, surgiu no contexto da Comunidade JINI o Projeto *Surrogate*.

Conforme citado em [Thompson, 2005], a principal deficiência coberta pela arquitetura *Surrogate* é a inabilidade nativa dos componentes descarregarem código, provavelmente devido à sua baixa capacidade computacional. Trata-se de uma arquitetura que suporta uma ampla gama de componentes com capacidades computacionais variadas, acomoda diversas tecnologias de conectividade e preserva a natureza *plug-and-play* provida pela tecnologia JINI.

A Figura 4.6 ilustra os principais termos definidos para a arquitetura *Surrogate*. Ela pressupõe a existência de uma máquina dotada de recursos computacionais que permitem a existência de um *surrogate host*, o qual conecta-se logicamente e fisicamente a um dispositivo por meio de um *interconnect*.

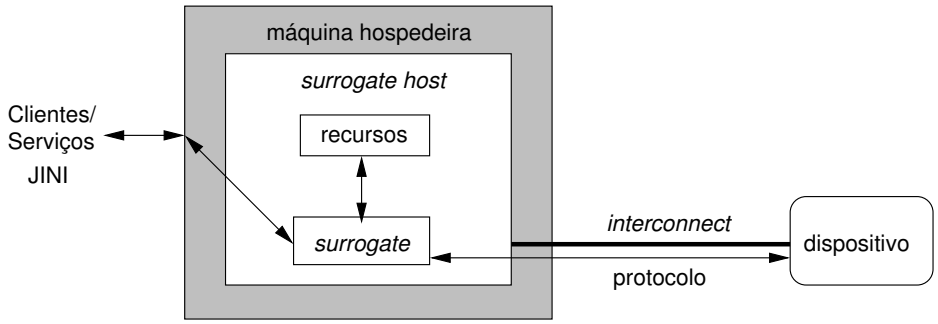

Figura 4.6: Arquitetura do Projeto *Surrogate*.

O *surrogate host* provê um ambiente de execução semelhante ao encontrado na J2SE, além das APIs JINI. Ele é especialmente projetado para carregar e executar *surrogates*. Um *surrogate* é a denominação pela qual é conhecido o código Java a ser executado no interesse de um determinado dispositivo.

Cada *surrogate* tem um ciclo de vida gerenciado por um *surrogate host*. Os estágios desse ciclo de vida são:

1. *descoberta do dispositivo*: nesse estágio, um *surrogate host* descobre a presença de um dispositivo (ou um dispositivo descobre a presença de um *surrogate host*);

2. *coleta do surrogate*: é a ação de localizar as Classes compiladas do *surrogate* e carregá-las para o *surrogate host*;

3. *ativação*: é o ato de instanciar-se o Objeto *surrogate* e fazer uma chamada de método para iniciar sua execução;

4. *execução*: é o estado estacionário de um *surrogate* em execução. Uma vez nesse estágio, *surrogates* podem utilizar os recursos providos pelo *surrogate host*, acessando e provendo Serviços JINI e comunicando-se com o dispositivo através de qualquer protocolo, incluindo-se nesse caso protocolos privados;

5. *desativação*: é o ato de encerrar a execução de um *surrogate*;

6. *eliminação*: é o ato de eliminar os Objetos associados com o *surrogate* desativado.

Os estágios de *descoberta do dispositivo* e *coleta do surrogate* são cobertos por um *protocolo de interconnect*, que depende especificamente da natureza do *interconnect* existente entre o *surrogate* e o *surrogate host*.

Uma vez que um *surrogate* é carregado para um *surrogate host*, um desses componentes monitora o caminho existente entre o *surrogate* e o dispositivo que ele representa. A perda desse caminho pode ser o resultado de condições normais ou de erro no *surrogate host*, no *surrogate*, no *interconnect* ou no dispositivo.

Se o dispositivo não é mais alcançável, seja devido a falhas no *interconnect* ou no próprio dispositivo, ou mesmo devido à desconexão ou desligamento do dispositivo, o *surrogate* que o representa precisa ser desativado de forma que os recursos a ele alocados pelo *surrogate host* possam ser retomados.

A especificação completa de uma Arquitetura *Surrogate* voltada para um determinado tipo de *interconnect* requer a especificação do protocolo utilizado naquele *interconnect* e das adições ao modelo de programação básico da Arquitetura que são decorrentes da utilização daquele *interconnect*.

Em [Sun Microsystems, 2001] é definido um protocolo de *interconnect* baseado em IP, juntamente com seu respectivo modelo de programação. Essa especificação define dois protocolos utilizados no primeiro estágio do ciclo de vida de um *surrogate*: são eles o MHAP (*Multicast Host Announcement Protocol*) e o MHRP (*Multicast Host Request Protocol*). O MHAP é utilizado pelo *surrogate host* para avisar a sua disponibilidade no *interconnect*. Já o MHRP é utilizado por um dispositivo para descobrir um *surrogate host* em uma localização desconhecida. Já o protocolo RRP (*Registration Request Protocol*) define as mensagens e os mecanismos utilizados pelo dispositivo para solicitar a um *surrogate host* que carregue os arquivos de Classe de seu respectivo *surrogate*, cobrindo assim o segundo estágio do ciclo de vida daquele *surrogate*.

Existem outros Subprojetos do Projeto *Surrogate*, focados em outros tipos de *interconnect*. Por exemplo, [Gemplus, 2002] propõe a definição de um *interconnect* para *smart cards*. Por outro lado, [Sun Microsystems, 2002] propõe estender a Arquitetura *Surrogate* para dispositivos que utilizam USB (*Universal Serial Bus*). Outro Subprojeto, ainda sob recente desenvolvimento e portanto sem especificações formais, busca definir um *interconnect* para dispositivos que utilizam o barramento serial IEEE-1394, também conhecido pelas denominações *Firewire* e *iLink*.

A incorporação do Projeto *Surrogate* no projeto de NMSs pode ser visualizada a partir da Figura 4.7. Ela ilustra dois Serviços, associados aos seus respectivos sistemas gerenciados. Um dos Serviços é associado a um *switch*, e o outro é executado como um *surrogate* associado a um PDA (*Personal Digital Assistant*) conectado a um *surrogate host* hospedado em uma estação de trabalho. Esses Serviços são gerenciados por uma Aplicação de Gerência executada como um *surrogate* associado a um telefone celular conectado também a um *surrogate host* hospedado em uma estação de trabalho. Os *surrogate hosts* citados poderiam ser instâncias de `madison`, a implementação de referência de um *surrogate host* provida pela Sun Microsystems.

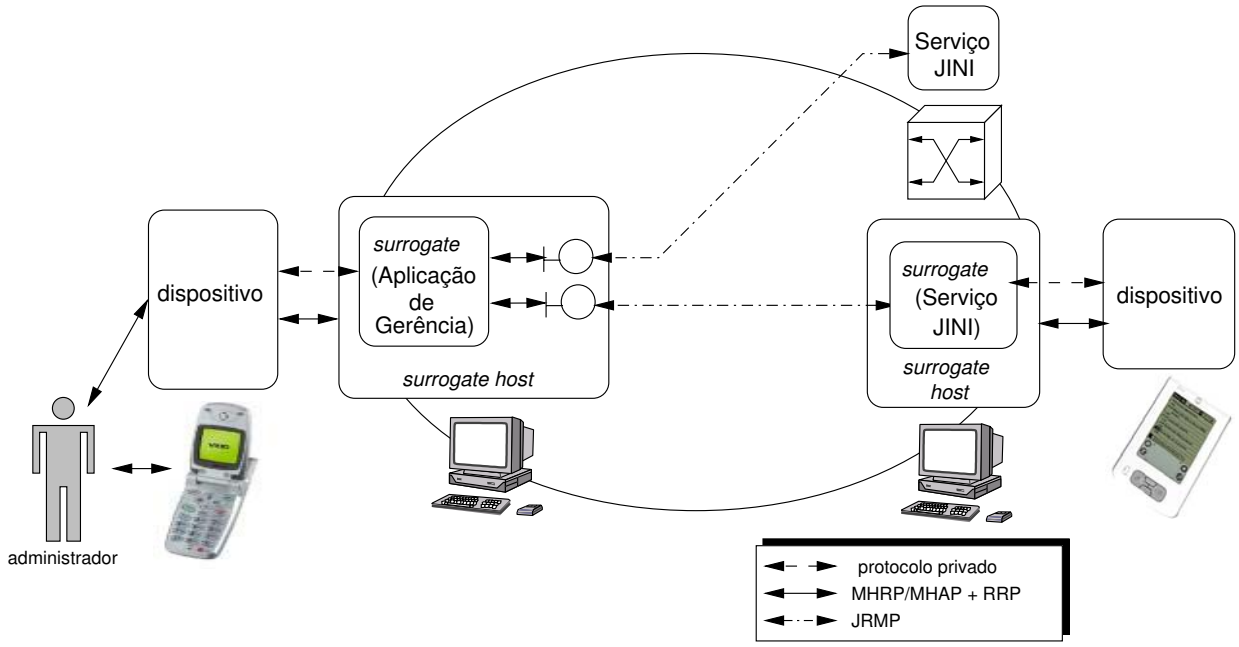

Figura 4.7: Incorporação do Projeto *Surrogate* no desenvolvimento de um NMS.

A Figura 4.7 ilustra como o Projeto *Surrogate* permite aos dispositivos com baixo poder computacional não só possuirem Aplicações de Gerência, mas também Serviços, associados. Naquela figura, o PDA é representado por um Serviço JINI associado, permitindo à Aplicação de Gerência executando no interesse do telefone celular a invocação remota de métodos junto àquele Serviço. No contexto da proposta descrita nesse capítulo, tais invocações consistem em operações de gerência.

Indo ainda mais fundo na aplicação do Projeto *Surrogate* no desenvolvimento de NMSs, pode-se também imaginar a integração daquele projeto com o Projeto *ServiceUI*, abordado na Seção 4.4. Essa integração é inicialmente sugerida em [Landis e Vasudevan, 2002], sendo ilustrada na Figura 4.8 a sua possível adaptação ao projeto de um NMS.

Na Figura 4.8, um protocolo baseado na XML (*eXtensible Markup Language*) [Bray et al., 1998] é utilizado na comunicação entre um *MIDlet* genérico e o Objeto de Interface com o Usuário. Este último recebe também a denominação de *Objeto Adaptador de Protocolo*, tendo-se em vista que ele adapta a interface oferecida pelo *proxy* a algum protocolo de rede que é compreendido pela Interface com o Usuário do dispositivo.

Existem abordagens alternativas para o emprego de dispositivos com baixo poder computacional utilizando-se JINI. Considere-se como exemplo a utilização de um *MIDlet* comunicando-se via HTTP com uma *servlet* instalada numa plataforma com um poder computacional relativamente alto. Considere-se enfim que essa *servlet* desempenhe o papel de uma Aplicação de Gerência ou de um Serviço JINI associado ao dispositivo.

Embora talvez seja a opção mais rápida de ser implementada para a inclusão de dispositivos com baixo poder computacional num NMS baseado em JINI, essa abordagem traz consigo alguns

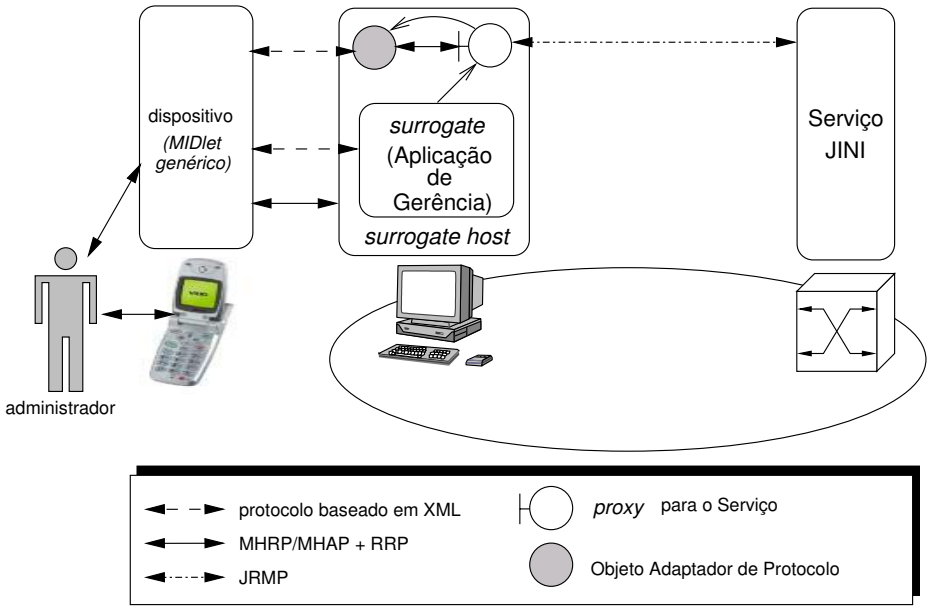

Figura 4.8: Integração dos Projetos *Surrogate* e *ServiceUI* no desenvolvimento de um NMS.

problemas potenciais. Esses problemas podem ser assim descritos:

- o *MIDlet* teria que conhecer *à priori* o endereço IP do Servidor HTTP onde reside a *servlet*, ou receber esta informação do usuário, esse último tendo que, ele próprio, conhecer aquela informação. Em suma, não haveria um mecanismo para descoberta dinâmica daquele Servidor, como é possibilitado pelo Projeto *Surrogate*;

- a comunicação entre o *MIDlet* e a *servlet*, por ser realizada através de HTTP (o único protocolo do nível de aplicação padrão suportado pela J2ME), não está salvaguardada por nenhum mecanismo de proteção eficaz que garanta autenticação e confidencialidade às mensagens trocadas entre aqueles componentes;

- no caso do *MIDlet* desempenhar o papel de uma Aplicação de Gerência, não haveria a possibilidade de incorporar as interfaces (gráficas) com o usuário necessárias à realização da gerência eficaz de todos os Serviços JINI presentes num determinado Domínio Gerenciado, assim como é possibilitado pelo Projeto *ServiceUI*.

Outra possível solução para esse caso seria a adoção de um protocolo do nível de aplicação próprio, baseado diretamente em *sockets*. O problema com essa abordagem é que não só o protocolo, mas sim todos os requisitos não-cumpridos pela abordagem anterior, deveriam ser implementados a partir do início. Conquanto essa possa de fato ser uma abordagem razoável para emprego a médio ou longo prazo, esforços no sentido da adoção do Projeto *Surrogate* e da implementação da incorporação a este projeto do projeto *ServiceUI* parecem ser mais rentáveis a curto prazo para a inclusão de dispositivos com baixo poder computacional em NMSs baseados em JINI.

4.8 Conclusão

Este Capítulo apresentou algumas considerações de âmbito geral sobre a utilização de JINI na construção de NMSs. Essas considerações podem ser assim sumarizadas:

1. a associação de Serviços JINI a sistemas gerenciados;
2. o agrupamento desses Serviços em Domínios Gerenciados;
3. a incorporação de Interfaces com o Usuários flexíveis nos *proxies* para esses Serviços;
4. a provisão de interações seguras entre Aplicações de Gerência e os Serviços;
5. uso dos mecanismos transacionais na comunicação entre Aplicações de Gerência e Serviços;
6. a inclusão de dispositivos com baixo poder computacional.

O próximo Capítulo descreve um NMS em cujo projeto foram aplicadas todas essas cinco considerações, constituindo-se, portanto, em uma das provas de conceitos desenvolvidas para sua validação.

Capítulo 5

Primeira prova de conceitos: a Baía de Dispositivos

5.1 Introdução

Em poucas palavras, pode-se definir a Baía de Dispositivos como um NMS no qual vários Serviços JINI associados a sistemas gerenciados (dispositivos) são agrupados num determinado Domínio (*baía*) e gerenciados por uma ou mais Aplicações de Gerência. Este capítulo descreve esta que é a primeira prova de conceitos referente às considerações apresentadas no capítulo anterior sobre a utilização da tecnologia JINI na construção de NMSs.

Este capítulo é organizado da seguinte forma: na Seção 5.2 é fornecida uma visão geral da Baía de Dispositivos, onde são introduzidos seus principais componentes: as Aplicações de Gerência e os Serviços JINI. A Seção 5.3 descreve quais os tipos de Serviços JINI desenvolvidos, enquanto a Seção 5.4 faz o mesmo com relação às Aplicações de Gerência. A Seção 5.5 define alguns cenários para operação destes componentes, analisando os resultados obtidos em cada um deles. A Seção 5.6 efetua as considerações finais sobre o sistema desenvolvido.

5.2 Visão geral do protótipo desenvolvido

A Figura 5.1 provê uma visão geral do protótipo desenvolvido para a Baía de Dispositivos. Conforme pode-se observar, esse protótipo possui dois Serviços JINI. Um deles é associado a um sistema gerenciado do tipo *desktop*, com capacidade de prover um ambiente de execução igual ou superior ao provido pela J2SE. O outro é associado a um sistema gerenciado da classe de um te-

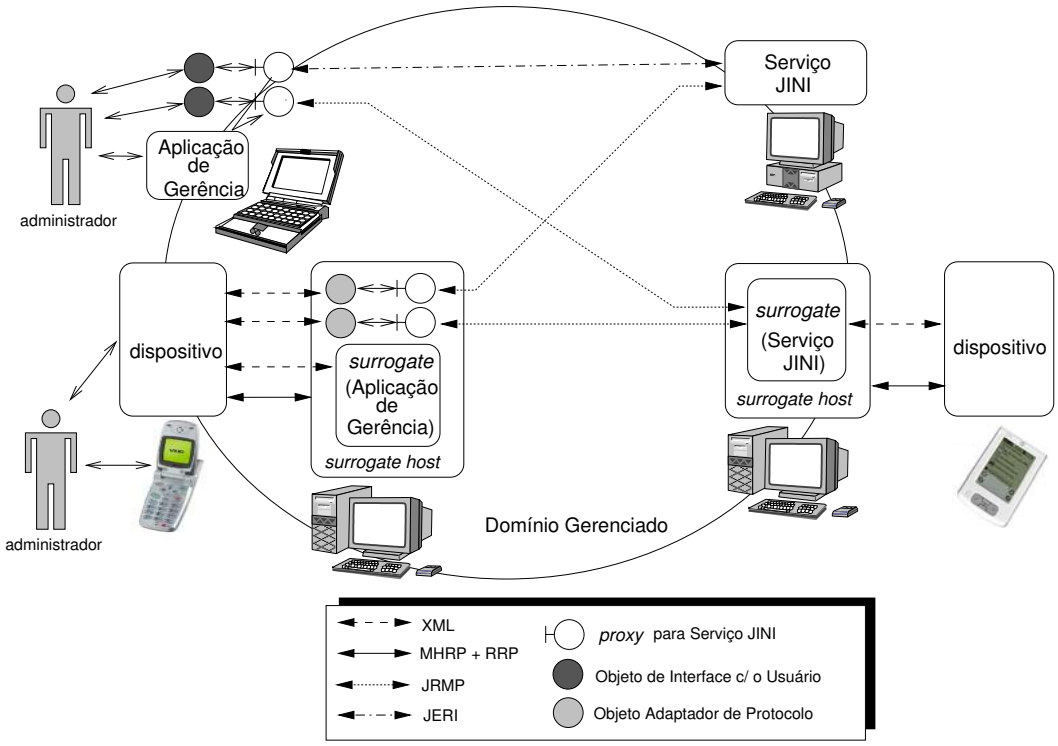

Figura 5.1: Visão geral da Baía de Dispositivos.

lefone celular ou um *palm-top*, por exemplo, capaz de prover um ambiente de execução restrito à CLDC (*Connected Limited Device Configuration*) da J2ME (*Java 2 Platform Micro Edition*). Esses Serviços são referidos aqui como Serviço *non-surrogate* e Serviço *surrogate*, respectivamente.

Similarmente, duas Aplicações de Gerência podem ser vistas na Figura 5.1. Uma delas é executada numa plataforma do tipo *desktop* e a outra é executada numa plataforma dotada de baixo poder computacional, tal qual um telefone celular ou um *palm top*, por exemplo. Essas Aplicações são referidas aqui como Aplicação de Gerência *non-surrogate* e Aplicação de Gerência *surrogate*, respectivamente.

Levando-se em conta as características de modelagem definidas na Seção 4.2, percebe-se que nesse NMS é empregado o menor nível de granularidade na modelagem dos Serviços JINI associados aos sistemas gerenciados: isto é, a cada sistema gerenciado está associado apenas um Serviço JINI.

Através de um LUS (não mostrado na Figura), os Serviços são agrupados numa Comunidade, que na Seção 4.3 é denominada de Domínio Gerenciado para propósitos de gerência de redes. As Aplicações de Gerência registram Ouvidores de Eventos junto àquele LUS, de forma a serem notificadas acerca de futuras inserções (ou mesmo eventuais remoções) de Serviços naquele Domínio Gerenciado. Conforme abordado na Seção 4.3, essa medida de projeto garante a manutenção de uma visão consistente dos Serviços atualmente presentes no Domínio em questão.

Outra consideração geral aplicada na construção da Baía de Dispositivos é a incorporação de Objetos de Interfaces com o Usuário nos próprios Serviços JINI, abordada inicialmente na Seção 4.4. Conforme descrito naquela Seção, essa medida de projeto provê flexibilidade ao NMS à medida em que ele pode prescindir do código necessário para a gerência amigável de um determinado sistema gerenciado.

Conforme ilustrado na Figura 5.1, o protótipo da Baía de Dispositivos contempla a utilização de dispositivos com baixo poder computacional, para tal fazendo uso do Projeto *Surrogate* como descrito inicialmente na Seção 4.7. Em particular, quando tais dispositivos assumem o papel de sistemas gerenciados, o Projeto *ServiceUI* é combinado com o Projeto *Surrogate*, de forma que os *proxies* para os Serviços associados a esses sistemas gerenciados contenham Objetos Adaptadores de Protocolo, além de Objetos de Interface com o Usuário.

Por fim, são também aplicadas as considerações feitas na Seção 4.5 e aplicadas, sempre que possível, restrições nos *proxies* para os Serviços. Neste caso, SSL é utilizado nas interações entre a Aplicação de Gerência e os Serviços JINI, de forma a prover autenticação dos Serviços e confidencialidade nas comunicações. Além disso, URLs HTTPMD são utilizadas de forma a garantir a integridade do código descarregado.

Um detalhe interessante é que, nos casos nos quais *surrogates* são associados a Aplicações de Gerência ou a Serviços JINI, as interações entre tais componentes não podem ser realizadas utilizando-se JERI, mas sim JRMP. Isso ocorre devido ao fato de que a implementação de *surrogate host* utilizada (`madison`) não suporta a versão 2.0 de JINI. Nesses casos, portanto, os mecanismos de segurança citados no parágrafo anterior não podem ser aplicados.

5.3 Serviços JINI desenvolvidos

Conforme citado na Seção anterior, foram desenvolvidas duas classes de Serviço JINI, sendo uma delas destinada à associação com dispositivos do tipo *desktop* (*non-surrogate*) e a outra destinada à associação com dispositivos com baixo poder computacional (*surrogate*). As subseções seguintes descrevem melhor estas classes de Serviço.

5.3.1 Serviço JINI *non-surrogate*

Um dos objetivos desse protótipo é mostrar que NMSs baseados em JINI podem incorporar sistemas legados baseados no binômio SMI/SNMP. Para tal, o Serviço JINI *non-surrogate* apresenta a arquitetura descrita na Figura 5.2.

O dispositivo ao qual associa-se um dos Serviços JINI desenvolvidos é um PC AMD Athlon operando na freqüência de 2 GHz e possuindo 1 GBytes de RAM. O sistema operacional em execução nesta máquina é a versão 10 da distribuição Linux Mandrake, e nela estão instaladas as versões 1.4.2_09 da J2SE e 2.0.2 do JTSK. Estas versões foram utilizadas para o desenvolvimento e utilização do Serviço descrito nessa subseção.

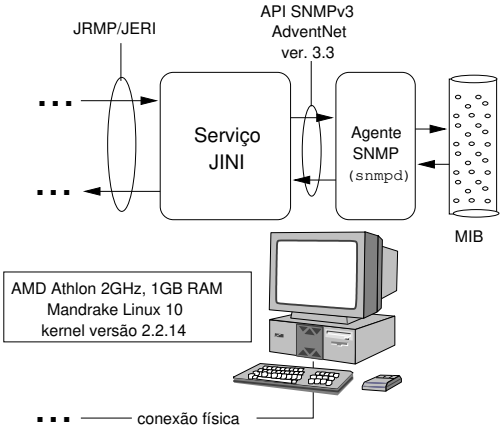

Figura 5.2: Arquitetura do Serviço *non-surrogate*.

Utilizando a versão 3.3 da API SNMPv3 disponibilizada em [AdventNet, 2005], as invocações remotas feitas ao Serviço são mapeadas para o envio de PDUs *Get Request* junto a um Agente SNMPv3 instalado no dispositivo. O Agente em questão é o `snmpd` (*snmp daemon*), incluso na versão 5.2 do *toolkit* NET-SNMP [Hardaker et al., 2005].

O `snmpd` é inicializado com base num arquivo de configuração previamente existente, e é capaz de suportar algumas MIBs. Uma delas é naturalmente a MIB-II [McCloghrie e Rose, 1991]. A Tabela 5.1 ilustra os MOs dessa MIB que são utilizados nesse protótipo, acompanhados de uma breve descrição dos mesmos. Todos eles pertencem ao grupo `system`.

MO	Descrição
`sysDescr`	descrição textual do sistema gerenciado.
`sysUpTime`	tempo (em centésimos de segundo) desde a última (re)inicialização do Sistema.
`sysContact`	nome do responsável pelo sistema gerenciado, juntamente com a informação de como contactar-se esse responsável.
`sysName`	nome dado administrativamente ao sistema gerenciado.
`sysLocation`	localização física do sistema gerenciado.

Tabela 5.1: MOs do grupo *system* da MIB-II utilizados.

A outra MIB suportada pelo `snmpd` e utilizada foi a MIB privada UCD-SNMP [Hardaker, 2005]. Essa MIB, a semelhança da MIB-II, comporta muitos MOs, sendo boa parte deles dispostos segundo tabelas. As tabelas de MOs da MIB UCD-SNMP utilizadas no protótipo são ilustradas pela Tabela 5.2, acompanhadas de uma breve descrição das mesmas.

Além dos MOs destas tabelas, alguns MOs escalares da MIB UCD-SNMP são utilizados. A Tabela 5.3 ilustra quais são esses MOs e fornece uma breve descrição dos mesmos.

Um exemplo de arquivo de configuração definido para o `snmpd` é ilustrado a seguir, tendo suas linhas sido numeradas apenas para propósitos didáticos.

Tabela	Descrição
`prTable`	contém informações sobre programas/*daemons* configurados para monitoração no arquivo de configuração do Agente.
`dskTable`	contém informações sobre a ocupação do disco da máquina. No arquivo de configuração do Agente são definidas as partições observadas e os valores de ocupação abaixo dos quais MOs de erro são configurados.
`laTable`	contém informações sobre a carga média do sistema gerenciado. No arquivo de configuração do Agente são definidos alguns períodos de observação e os valores acima dos quais MOs de erro são configurados.
`fileTable`	contém informações sobre arquivos monitorados. No arquivo de configuração do Agente são definidos os arquivos a serem monitorados e valores de tamanho acima dos quais MOs de erro são configurados.

Tabela 5.2: Tabelas de MOs da MIB UCD-SNMP utilizadas.

MO	Descrição
`memTotalSwap`	memória *swap* total configurada para o *host*.
`memAvailSwap`	memória *swap* disponível no *host*.
`memTotalReal`	memória real (física) total no *host*.
`memAvailReal`	memória real (física) disponível no *host*.
`memTotalFree`	memória total disponível no *host*.
`memMinimumSwap`	valor de memória *swap* abaixo do qual MOs de erro são setados.
`ssCpuRawUser`	tempo de CPU do usuário.
`ssCpuRawSystem`	tempo de CPU do sistema.
`ssCpuRawIdle`	tempo *idle* de CPU.

Tabela 5.3: MOs escalares da MIB UCD-SNMP utilizados.

```
01  createUser  helcio  MD5  bravosierra  DES  mikesierra
02  group  bant  usm  helcio
03  view  all  included  .1  80
04  access  bant  ""  usm  authpriv  exact  all  all  all
05  proc  telnetd
06  proc  httpd2      4
07  proc  ftpd
08  disk  /            5%
09  disk  /mnt/win_c  10%
10  disk  /mnt/win_d  15%
11  load  0.5
12  file  /home/helcio/applications.zip  10000
13  file  /home/helcio/poseidon.zip  50000
14  file  /home/helcio/notes.zip  100000
```

Este arquivo possui diretivas que permitem configurar, em tempo de inicialização, certos MOs

nas MIBs SNMPv3 VACM e USM, suportadas pelo Agente representado pelo `snmpd`.

Na linha 01, a diretiva `createUse` cria o usuário *helcio*, e estabelece como protocolos de autenticação e encriptação o MD5 e o DES, respectivamente. As senhas de autenticação e encriptação definidas para esse Usuário são *bravosierra* e *mikesierra*, respectivamente.

Na linha 02, o usuário *helcio* é associado ao grupo *bant* mediante aplicação da diretiva `group`. Na linha 03, a diretiva `view` é utilizada para criação de uma Visão de MIB denominada *all*, que contempla todos os MOs suportados pelo Agente.

Na linha 04, mediante aplicação da diretiva `access`, é dado direito aos usuários pertencentes a *bant* a leitura e escrita dos MOs definidos em *all*, bem como a notificações de mudanças verificadas naqueles MOs.

Assim, de acordo com os três parágrafos anteriores, o Agente é configurado de forma que o usuário *helcio* (mediante aplicação de autenticação e privacidade) tem direito de leitura e escrita de todos os MOs suportados pelo Agente, bem como o direito de ser notificado sobre mudanças observadas naqueles MOs.

Nas linhas 05, 06 e 07, a diretiva `proc` é utilizada para checar se os processos *telnetd*, *httpd2* e *ftpd* encontram-se em execução na máquina. A aplicação da diretiva `proc` faz com que sejam configurados MOs na tabela `prTable`. No caso de haver mais de quatro processos *http2* em execução, MOs de erro são também criados nessa tabela.

Nas linhas 08, 09 e 10, a aplicação da diretiva `disk` faz com que sejam configurados MOs na tabela `diskTable` para obter-se o valor do espaço disponível nas partições */*, */mnt/win_c* e */mnt/win_d* da máquina, verificando se estes valores encontram-se inferiores a *5%*, *10%* e *15%* de suas capacidades totais, respectivamente.

Na linha 11, a aplicação da diretiva `load` faz com que sejam configurados MOs na tabela `laTable` para obter-se a carga do sistema gerenciado, e verificar se este valor excede os 50% de seu valor máximo em intervalos de 01 (um), 05 (cinco) e 15 (quinze) minutos, respectivamente.

Nas linhas 12, 13 e 14, a aplicação da diretiva `file` faz com que sejam configurados MOs na tabela `fileTable` para obter-se o tamanho dos arquivos *applications.zip*, *poseidon.zip* e *notes.zip*, todos eles localizados no diretório */home/helcio* do sistema gerenciado. Também verifica-se se estes arquivos possuem tamanhos superiores a 10000 kB, 50000 kB e 100000 kB, respectivamente.

O Serviço *non-surrogate* implementa a interface `DesktopService`, descrita a seguir:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface DesktopService extends Remote {
   public GeneralInfo getGeneralInfo() throws RemoteException;
   public CpuAndMemoryInfo getCpuAndMemoryInfo() throws RemoteException;
   public FilesAndDiskInfo getFilesAndDiskInfo() throws RemoteException;
   public LoadAndProcesses getLoadAndProcesses() throws RemoteException;
}
```

Os Objetos `GeneralInfo`, `CpuAndMemoryInfo`, `FilesAndDiskInfo` e `LoadAndProcessesInfo` funcionam como contâineres para as informações relacionadas aos MOs mencionados nas Tabelas

anteriormente ilustradas. Mais precisamente, o Objeto `GeneralInfo` armazena informações relacionadas aos MOs mencionados na Tabela 5.1. O Objeto `CpuAndMemoryInfo` armazena informações relacionadas aos MOs mencionados na Tabela 5.3. O Objeto `FileAndDiskInfo` contém informações relacionadas aos MOs presentes nas tabelas `fileTable` e `dskTable`, mencionadas na Tabela 5.2. Por fim, o Objeto `LoadAndProcessesInfo` armazena informações relacionadas aos MOs presentes nas tabelas `laTable` e `prTable`, mencionadas também na Tabela 5.2.

O *proxy* para esse Serviço contém três atributos: o nome administrativo do Serviço, um Objeto de Interface com o Usuário e um Objeto Adaptador de Protocolo. O nome administrativo é um Objeto da Classe `net.jini.lookup.entry.Name`, e os Objetos de Interface com o Usuário e Adaptador de Protocolo são obtidos a partir de Objetos da Classe `net.jini.lookup.entry.UIDescriptor`. O Objeto de Interface do Usuário destina-se à execução no espaço de endereçamento da Aplicação de Gerência executada num dispositivo do tipo *desktop*, enquanto o Objeto Adaptador de Protocolo é executado no espaço de endereçamento do *surrogate host* que hospeda o *surrogate* (Aplicação de Gerência) associado a um dispositivo com baixo poder computacional.

No Projeto da GUI para o Serviço JINI desenvolvido, a API JAAS (*Java Authentication and Authorization Service*) foi utilizada de forma que apenas usuários devidamente identificados possam inicializá-lo.

A Figura 5.3 ilustra esta GUI, notadamente após sua inicialização. Construída com base na API Java *swing*, ela contém três painéis. No painel destinado à configuração da comunicação com o Agente SNMP, são estabelecidos os parâmetros utilizados naquela comunicação: o endereço IP do Agente, o *port* UDP utilizado pelo Agente para a recepção de mensagens, a definição do tempo máximo de espera de uma mensagem de resposta antes do reenvio da solicitação (*timeout*) e o número máximo de retransmissões (*retries*). São também definidos naquele painel os parâmetros de segurança SNMPv3 utilizados naquela comunicação: o nome do usuário em cujo interesse a mensagem está sendo enviada, o protocolo de autenticação utilizado (opções disponíveis são o MD5 e o SHA), e as senhas de autenticação e privacidade a serem utilizadas para a geração de suas respectivas chaves.

No painel destinado à configuração dos parâmetros JINI do Serviço, define-se o nome atribuído administrativamente ao Serviço, o grupo (Domínio Gerenciado) ao qual o Serviço adere, o tipo de mecanismo utilizado pelo Serviço para descoberta do LUS e o tipo de renovação de *lease* utilizado junto àquele LUS para o registro do *proxy* para o Serviço. Caso o mecanismo *unicast* seja escolhido para a descoberta do LUS, campos destinados ao fornecimento do endereço IP e do *port* TCP utilizados pelo LUS são habilitados e aquelas informações são fornecidas. De maneira análoga, caso o mecanismo de renovação do *lease* associado ao Serviço seja controlado pelo usuário (não sendo realizado de maneira automática), são habilitados os campos destinados ao fornecimento do período de renovação do *lease* e do período máximo ao fim do qual esta renovação deixa de ser feita.

O último dos painéis destina-se à exibição de mensagens alusivas tanto a procedimentos internos realizados pelo Serviço quanto à comunicação com os demais componentes do NMS. Por fim, a GUI contém ainda botões destinados à inicialização do Serviço, à exibição de uma janela do tipo *about* contendo informações básicas sobre ele, e ao seu encerramento. As funcionalidades destes botões

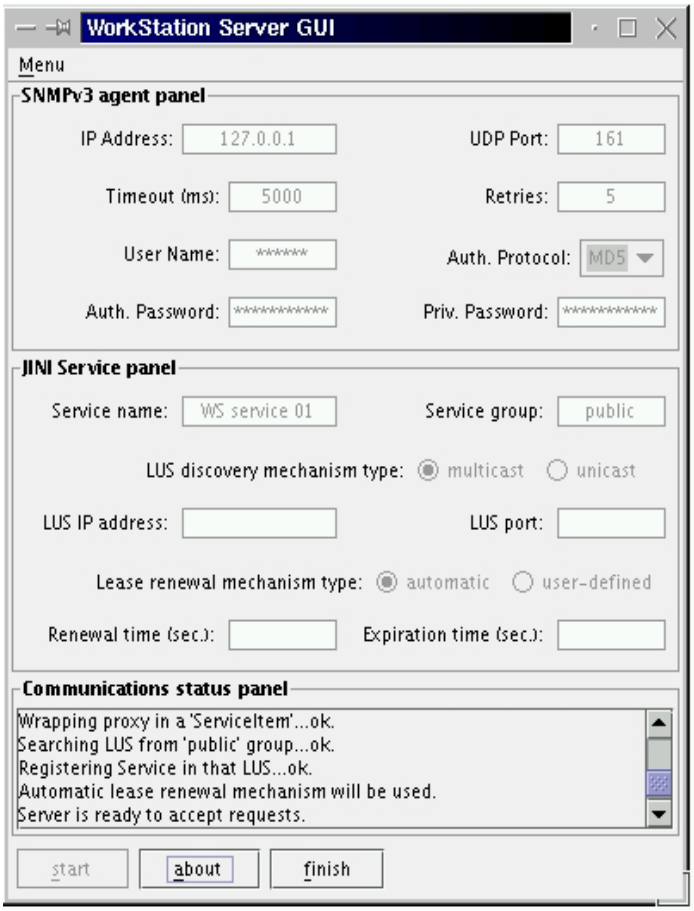

Figura 5.3: GUI desenvolvida para o Serviço *non-surrogate*.

encontram-se replicadas em ítens de *Menu*.

5.3.2 Serviço JINI *surrogate*

Enquanto o Serviço JINI *non-surrogate* mantém compatibilidade com sistemas legados baseados no binômio SMI/SNMP, o projeto do Serviço JINI *surrogate* não possui este tipo de enfoque. De fato, conforme ilustra a Figura 5.4, o enfoque deste projeto são os protocolos utilizados na comunicação entre os seus componentes.

Em particular, na comunicação entre o dispositivo e o *surrogate host* são utilizados os protocolos MHRP e RRP, e nas comunicações entre o dispositivo e o seu *surrogate* é utilizado um protocolo simples baseado em XML contendo mensagens do tipo *request*/*response*.

O desenvolvimento do dispositivo baseou-se na implementação realizada por Keith Thompson de um dispositivo CLDC que implementa o protocolo de *interconnect* definido para redes IP, confor-

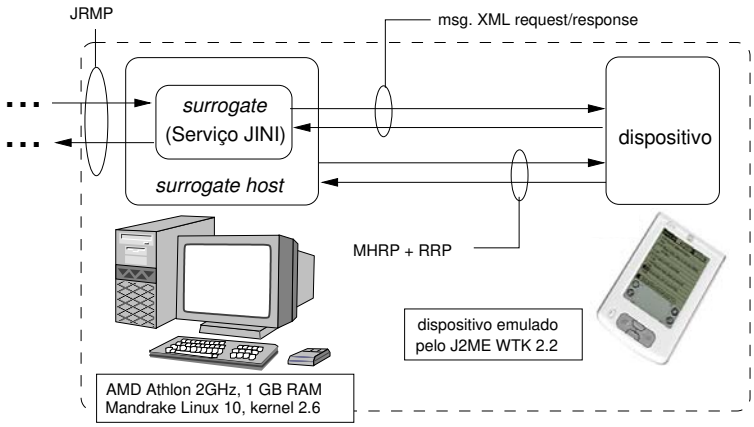

Figura 5.4: Arquitetura do Serviço *surrogate*.

me descrito em [Sun Microsystems, 2001]. O código daquela implementação é baseado na versão 1.0.3 da kVM, a JVM utilizada para a configuração CLDC da J2ME. Algumas restrições impostas pela configuração CLDC àquele dispositivo são enumeradas a seguir:

- o GCF (*Generic Connection Framework*) da CLDC não provê um mecanismo para recepção de mensagens UDP *multicast*. Portanto, o código não implementa o MHAP. O dispositivo implementa apenas o MHRP;
- o GCF não permite a modificação do parâmetro TTL (*Time-To-Live*) para mensagens *multicast*;
- a CLDC não permite a obtenção interna do endereço IP do próprio dispositivo. Desde que os protocolos de descoberta IP requerem que este endereço esteja nas mensagens, ele deve ser provido externamente.

Mesmo com as limitações acima mencionadas, o código desenvolvido descobre um *surrogate host*, registra com ele um *surrogate* e mantém uma monitoração constante do *status* daquele *surrogate*. O *surrogate*, por sua vez, tenta registrar um Serviço simples junto ao LUS.

Este código foi adaptado na construção de um *MIDlet* baseado na especificação MIDP (*Mobile Information Device Profile*) 1.0. Conforme ilustrado pela Figura 5.4, este *MIDlet* é executado em um dispositivo emulado pela versão 2.2 do WTK (*Wireless Toolkit*), que contém a J2ME distribuída pela Sun Microsystems.

O *MIDlet* construído possui as mesmas restrições enumeradas há pouco para o dispositivo CLDC, e seu *surrogate* contém o código de um Serviço que implementa a interface `LCPDService`, descrita a seguir.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
public interface LCPDService extends Remote {
   public String[] getGeneralInfo() throws RemoteException;
}
```

Conforme observado, a interface `LCPDService` é bem mais simples que a interface `DesktopService`, implementada pelo Serviço *non-surrogate*. O arranjo de `String` retornada contém apenas algumas informações básicas sobre o dispositivo: uma descrição textual, um nome administrativo, a identificação do proprietário e o valor do tempo decorrido desde a última reinicialização do dispositivo.

Similarmente ao Serviço *non-surrogate*, o Serviço *surrogate* possui como seus atributos um Objeto da Classe `net.jini.lookup.entry.Name` e dois Objetos da Classe `net.jini.lookup.entry.UIDescriptor`. Esses Objetos cumprem as mesmas funções definidas para o Serviço anterior, ou seja: o Objeto da Classe `net.jini.lookup.entry.Name` fornece o nome administrativo do Serviço e os Objetos da Classe `net.jini.lookup.entry.UIDescriptor` provêem, em última instância, os Objetos de Interface com o Usuário e Adaptador de Protocolo.

A plataforma de hardware/software na qual é executado o *surrogate host* (`madison`) é muito semelhante àquela descrita na subseção anterior para o Serviço associado a dispositivos do tipo *desktop*. A única diferença é que essa nova plataforma contém também as APIs referentes à Arquitetura *Surrogate* e à especificação do *interconnect* para redes IP.

5.4 Aplicações de Gerência desenvolvidas

Conforme citado na Seção 5.2, foram desenvolvidas duas classes de Aplicações de Gerência, sendo uma delas destinada à associação com dispositivos do tipo *desktop* (*non-surrogate*) e a outra destinada à associação com dispositivos com baixo poder computacional (*surrogate*). As subseções seguintes descrevem melhor essas duas classes.

5.4.1 Aplicação de Gerência *non-surrogate*

A Figura 5.5 ilustra a arquitetura da Aplicação de Gerência desenvolvida para dispositivos do tipo *desktop*. As plataformas de hardware e software para esse componente são idênticas às descritas na Subseção 5.3.1 para o Serviço JINI associado a dispositivos do tipo *desktop*, ou seja: a Aplicação de Gerência é executada num PC Athlon operando na freqüência de 2 GHz e possuindo 1 GBytes de RAM, o sistema operacional em execução nesta máquina é a versão 10 da distribuição Linux Mandrake, e nela encontram-se instaladas as versões 1.4.2_09 da J2SE e 2.0.2 do JTSK, necessárias ao desenvolvimento e utilização da Aplicação de Gerência.

Similarmente ao que foi dito para o Serviço JINI na Subseção 5.3.1, a API JAAS foi utilizada no projeto da GUI para esta Aplicação de Gerência de forma que apenas usuários devidamente identificados possam inicializá-la.

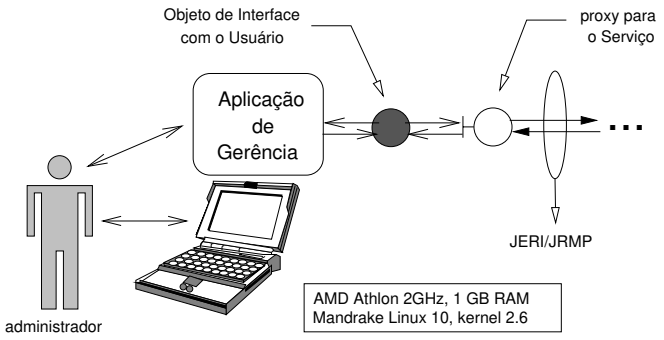

Figura 5.5: Arquitetura da Aplicação de Gerência *non-surrogate*.

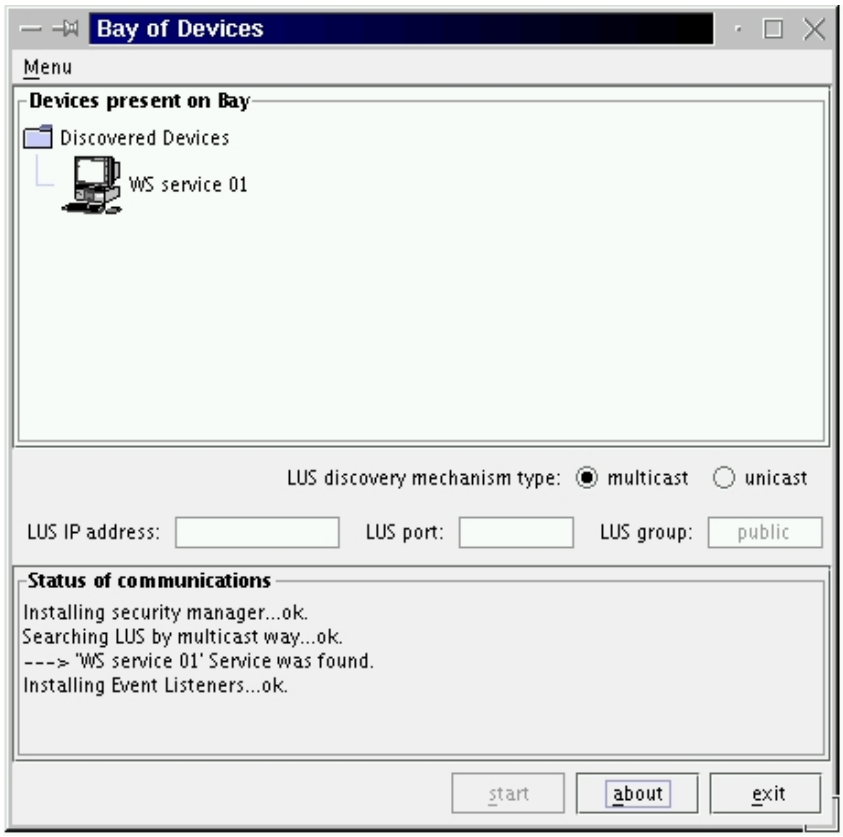

Figura 5.6: GUI desenvolvida para a Aplicação de Gerência *non-surrogate*.

A Figura 5.6 ilustra essa GUI, também construída a partir da API Java *swing*, notadamente após a inicialização da Aplicação de Gerência. Essa inicialização inclui a escolha do tipo de mecanismo utilizado para a descoberta do LUS e o Domínio Gerenciado de responsabilidade daquele componente. De maneira análoga ao observado na GUI para o Serviço JINI descrito na Seção 5.3.1, caso

o tipo de descoberta *unicast* seja selecionado, os campos destinados à recepção das informações relacionadas ao endereço IP e ao *port* TCP utilizados pelo LUS são habilitados, e tais informações devem ser fornecidas.

Durante a inicialização da Aplicação de Gerência, após o LUS ser descoberto é procedida uma consulta junto àquele Serviço sobre a existência de possíveis Serviços já presentes no Domínio Gerenciado sob sua responsabilidade. Na Figura 5.6, por exemplo, o Serviço `WS service 01` já encontrava-se presente no Domínio Gerenciado `public`. Esse Serviço, juntamente com outros Serviços que irão possivelmente entrar naquele Domínio, é representado por um ícone localizado no painel principal da GUI. Esse painel contém uma estrutura em árvore destinada a acolher tais representações. Num ambiente repleto de sistemas gerenciados (dispositivos) possuindo Serviços JINI associados, esse painel encontra-se repleto de ícones, consistindo realmente numa baía de dispositivos.

Prosseguindo na inicialização da Aplicação de Gerência, é instalado um Ouvidor de Evento junto ao LUS, de forma que este componente é informado acerca de possíveis entradas ou saídas de Serviços no Domínio Gerenciado. Essas entradas ou saídas são refletidas no painel principal da Aplicação de Gerência, que passa então a exibir também os ícones alusivos ao Serviços recém-ingressos e a deixar de exibir os ícones alusivos aos Serviços recém-egressos do Domínio Gerenciado sob responsabilidade do LUS.

5.4.2 Aplicação de Gerência *surrogate*

Em termos arquiteturais, o projeto da Aplicação de Gerência associada a dispositivos com baixo poder computacional guarda muita semelhança com o projeto do Serviço associado a dispositivos de mesma natureza. De fato, conforme ilustrado pela Figura 5.7, as plataformas de hardware/software são idênticas em ambos os projetos e os protocolos utilizados nas comunicações entre seus componentes seguem a mesma natureza.

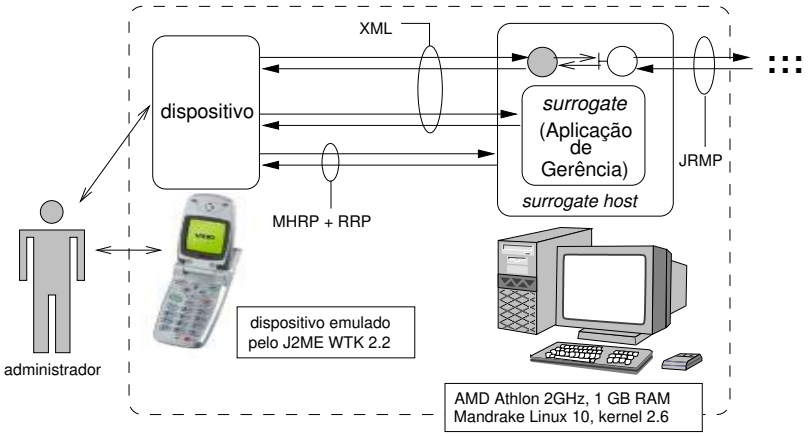

Figura 5.7: Arquitetura da Aplicação de Gerência *surrogate*.

Neste projeto, o dispositivo (que também foi baseado no dispositivo CLDC implementado por Keith Thompson) descobre o *surrogate host* utilizando o MHRP e registra com ele o seu *surrogate* utilizando o RRP, de maneira análoga ao que foi descrito para o Serviço JINI na Subseção 5.3.2. Porém, haja vista que neste caso o *surrogate* implementa a lógica da Aplicação de Gerência, seu comportamento após sua ativação assemelha-se ao comportamento descrito na subseção anterior para a versão deste componente associada a dispositivos do tipo *desktop*.

Ao ser ativado, o *surrogate* contacta o LUS e descobre quais os Serviços presentes no Domínio Gerenciado sob responsabilidade daquele componente. Em seguida, o *surrogate* passa a comunicar-se com o dispositivo utilizando um protocolo baseado em XML. Esse protocolo prevê, seqüencialmente, o fornecimento de uma mensagem contendo a lista de Serviços ao dispositivo e a recepção de outra mensagem contendo a escolha de um dentre aqueles Serviços. Essa escolha é feita pelo usuário da Aplicação de Gerência observando o *display* do dispositivo.

Quando o *surrogate* obtém a identificação do Serviço escolhido, ele obtém o Objeto Adaptador de Protocolo a partir do *proxy* para aquele Serviço e delega àquele Objeto toda a responsabilidade pela comunicação com o dispositivo. Essa comunicação é realizada também mediante o emprego de um protocolo baseado em XML. Esse protocolo prevê, seqüencialmente, uma mensagem contendo as opções disponibilizadas pelo Serviço, uma mensagem contendo a escolha de uma dentre aquelas opções e uma mensagem contendo os dados relacionados à opção escolhida.

5.5 Resultados experimentais

De forma a melhor analisar o NMS desenvolvido, inicialmente são descritos quatro cenários de operação, ilustrados na Tabela 5.4. Logo após a descrição destes cenários é realizada análise dos resultados obtidos a partir de cada um deles, confrontando-os entre si.

5.5.1 Primeiro cenário de operação

Conforme descrito na Tabela 5.4, o primeiro cenário de operação concebido diz respeito à comunicação entre a Aplicação de Gerência *non-surrogate* e o Serviço *non-surrogate*. Conquanto esse cenário, ilustrado pela Figura 5.8, não envolva componentes que fazem uso do Projeto *Surrogate*, ele é útil para mostrar como NMSs baseados em JINI podem absorver sistemas legados baseados no binômio SMI/SNMP.

De acordo com o que foi descrito na Subseção 5.4.1, durante a inicialização da Aplicação de Gerência ela descobre um LUS responsável por um determinado Domínio Gerenciado e busca junto a este componente determinados tipos de Serviços que pertençam àquele Domínio. Independentemente de existirem ou não Serviços já presentes naquele Domínio, um Ouvidor de Eventos é instalado junto ao LUS de forma a permitir à Aplicação de Gerência ser notificada acerca da entrada ou saída de Serviços no Domínio. Estes passos, além daqueles relacionados à inicialização dos Serviços, não são cobertos pela Figura 5.8.

primeiro cenário: Aplicação de Gerência *non-surrogate* & Serviço *non-surrogate*.	segundo cenário: Aplicação de Gerência *surrogate* & Serviço *non-surrogate*.
terceiro cenário: Aplicação de Gerência *non-surrogate* & Serviço *surrogate*.	quarto cenário: Aplicação de Gerência *surrogate* & Serviço *surrogate*.

Tabela 5.4: Cenários de aplicação concebidos.

Quando um usuário (administrador de rede) seleciona um dos Serviços presentes no painel da Aplicação de Gerência (os quais são referenciados de acordo com o ilustrado pela Figura 5.6), o código do Objeto de Interface com o Usuário para aquele Serviço é obtido a partir do seu respectivo *proxy*. Esse Objeto é instanciado, e a Interface é exibida (a). O usuário a utiliza então para comunicar-se com o Serviço e assim gerenciar o dispositivo ao qual este Serviço é associado (b).

5.5.2 Segundo cenário de operação

A Figura 5.9 ilustra um exemplo da aplicação do segundo cenário de operação concebido para a Baía de Dispositivos. Esse cenário assume a utilização do Projeto *Surrogate* apenas no projeto da Aplicação de Gerência. Inicialmente, o dispositivo descobre o *surrogate host* utilizando o MHRP e registra seu respectivo *surrogate* (Aplicação de Gerência) utilizando o RRP.

Após o seu registro e ativação, o *surrogate* contacta o LUS em busca de *proxies* para os Serviços pertencentes ao Domínio Gerenciado sob responsabilidade deste LUS. Em seguida, o *surrogate* armazena aquelas referências internamente. Esses procedimentos foram descritos na Subseção 5.4.2, não sendo cobertos pela Figura 5.9.

Na Figura 5.9, o *surrogate* envia uma mensagem XML `listOfServices` para o dispositivo, contendo uma lista dos Serviços disponíveis naquele Domínio Gerenciado. Ao receber essa mensagem, o dispositivo ajusta o seu *display* para a exibição daquela lista utilizando componentes de

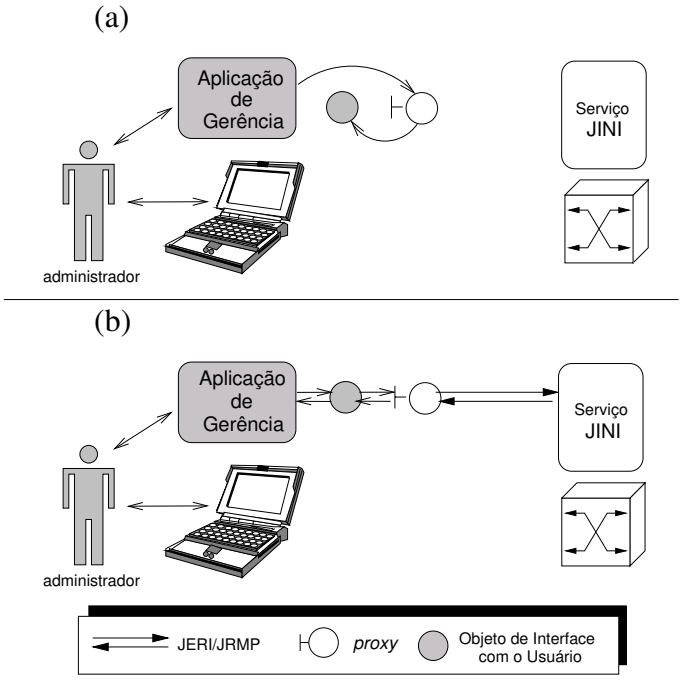

Figura 5.8: Primeiro cenário de operação concebido.

Interface com o Usuário. Quando o usuário escolhe um daqueles Serviços, uma mensagem XML `serviceChosen` é enviada para o *surrogate host* (a).

Após a recepção da mensagem `serviceChosen`, o *surrogate* descarrega o código do Objeto Adaptador de Protocolo a partir do *proxy* para o Serviço escolhido. Esse Objeto envia uma mensagem XML `listOfOptions` para o dispositivo, descrevendo as opções disponibilizadas pelo seu Serviço associado. Essa mensagem é interpretada pelo dispositivo, que ajusta o seu *display* novamente de forma a exibir estas opções. Quando o usuário escolhe uma delas, uma mensagem XML `optionChosen` contendo a opção escolhida é enviada para o Objeto Adaptador de Protocolo (b) .

Após receber a mensagem `optionChosen`, o Objeto Adaptador de Protocolo invoca o método relacionado à opção escolhida. Essa invocação é realizada utilizando-se JRMP, e seu resultado é posto numa mensagem XML `listOfData`. Essa mensagem é enviada para o dispositivo, que a recebe e ajusta o seu *display* para exibir aqueles resultados (c).

Conforme pode ser observado na Figura 5.9, as mensagens `listOfOptions`, `optionChosen` e `listOfData` identificam o Serviço ao qual são relacionadas. A mensagem `listOfData` identifica ainda a opção a qual seus dados correspondem.

5.5.3 Terceiro cenário de operação

A Figura 5.10 ilustra um exemplo da aplicação do terceiro cenário de operação concebido para a Baía de Dispositivos. Esse cenário assume a utilização do Projeto *Surrogate* apenas no projeto do

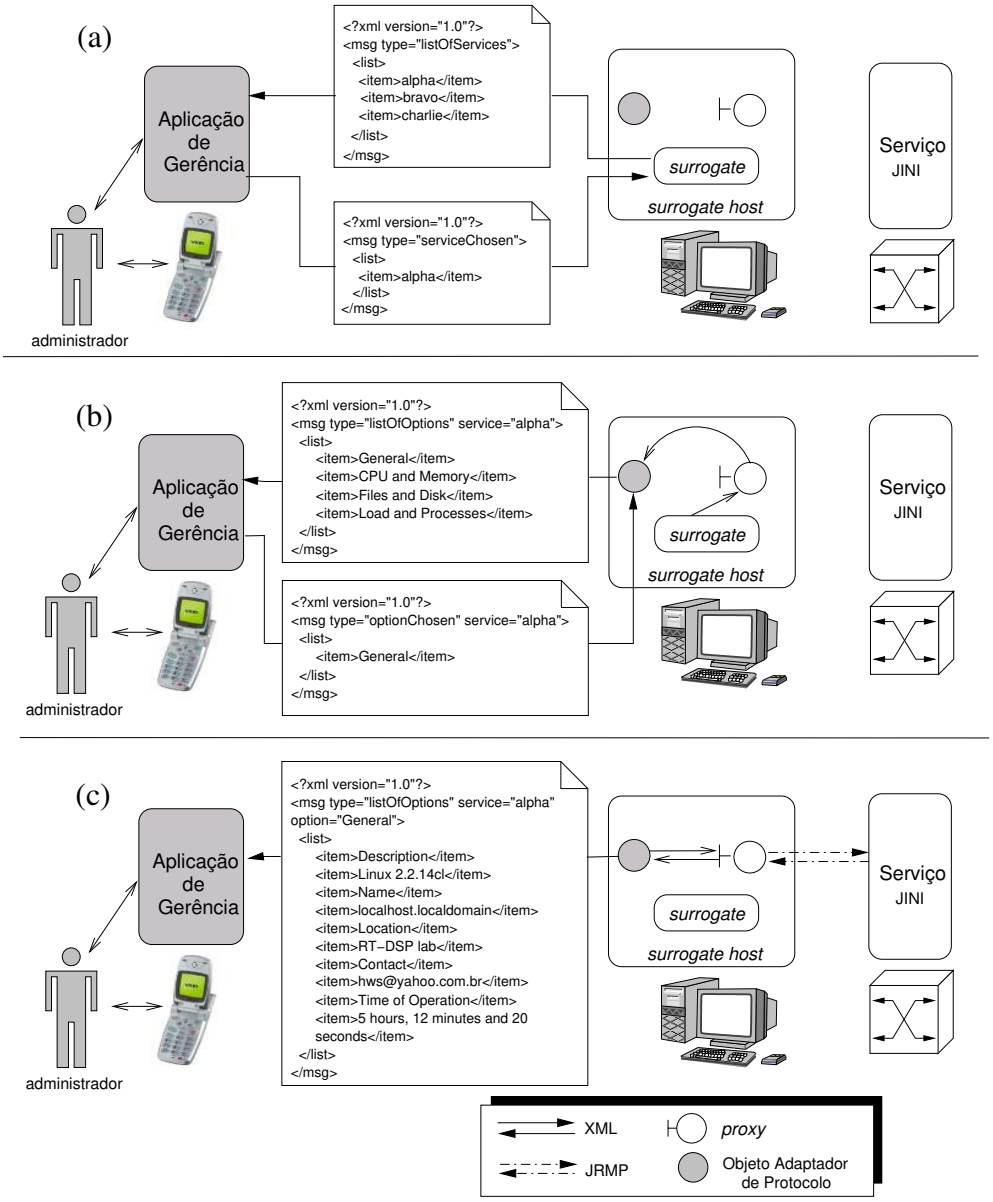

Figura 5.9: Segundo cenário de operação concebido.

Serviço JINI.

Durante seu processo de inicialização, o dispositivo descobre o *surrogate host* utilizando o MHRP e registra junto àquele componente um *surrogate* encapsulando o código do Serviço e do seu *proxy* relacionado utilizando o RRP. Uma vez carregado e ativado, o *surrogate* instancia o Serviço, descobre um determinado LUS e registra o *proxy* junto a este componente. Estes procedimentos não são cobertos pela Figura 5.10, sendo descritos na Subseção 5.3.2.

Quando o usuário seleciona aquele Serviço no painel da Aplicação de Gerência, o código do

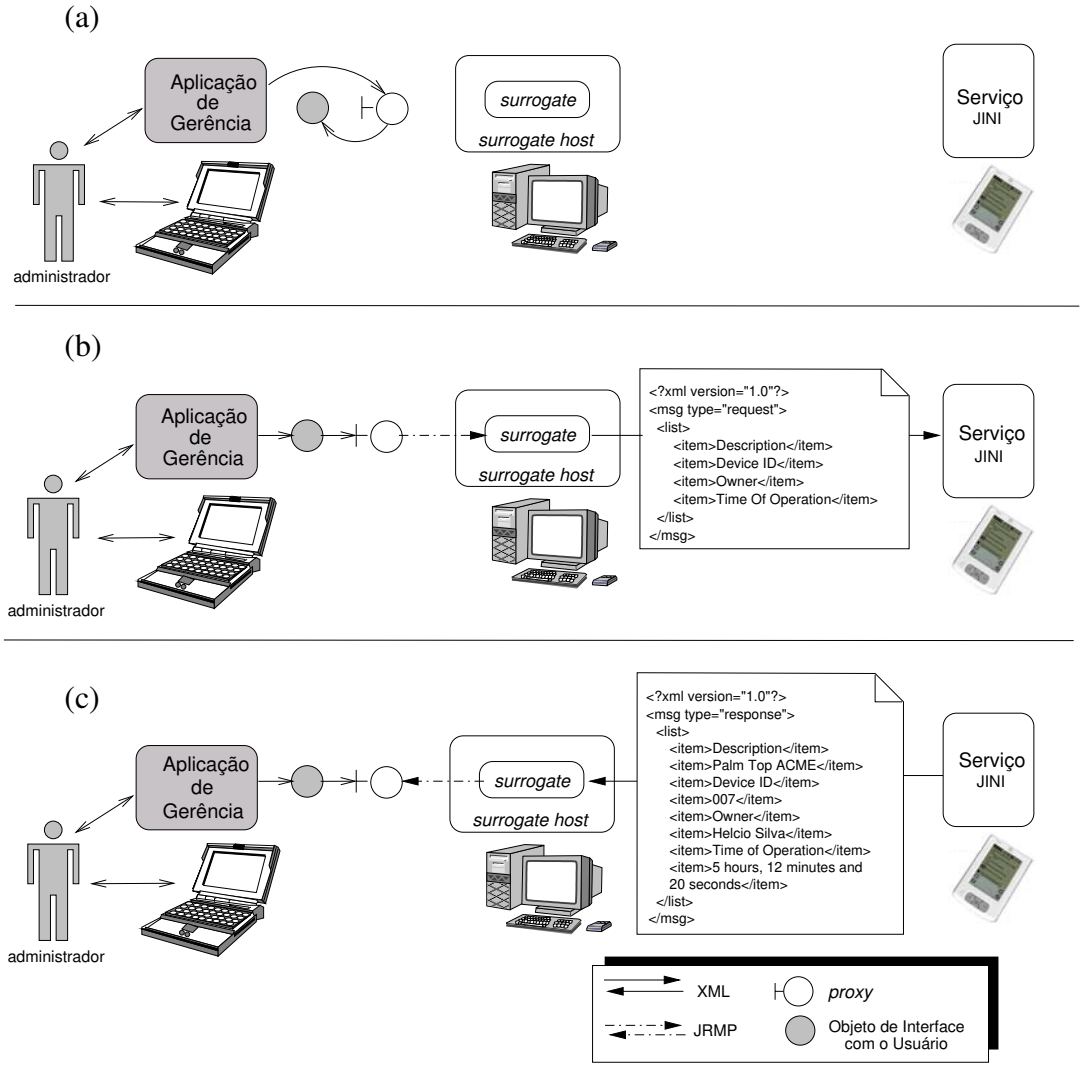

Figura 5.10: Terceiro cenário de operação concebido.

seu Objeto de Interface com o Usuário é obtido a partir do seu *proxy* e instanciado. O usuário utiliza a Interface e invoca indiretamente os métodos no *proxy* para o Serviço. Estas invocações são realizadas utilizando o JRMP, e endereçam o *surrogate* responsável pela execução da lógica do Serviço. O *surrogate* comunica-se então com o dispositivo utilizando o protocolo *request/response* descrito na Subseção 5.3.2 (b).

Após a recepção e processamento da mensagem `request`, o dispositivo coleta as informações solicitadas e as encapsula numa mensagem `response`. Essa última mensagem é enviada de volta para o *surrogate*, que a interpreta, extrai as informações e a encaminha para o *proxy* utilizando o JRMP. As informações são finalmente formatadas e apresentadas na Interface com o Usuário para aquele Serviço (c).

5.5.4 Quarto cenário de operação

A Figura 5.11 ilustra um exemplo da aplicação do quarto cenário de operação concebido para a Baía de Dispositivos. Esse cenário assume a utilização do Projeto *Surrogate* nos dois lados: no projeto do Serviço JINI e no projeto da Aplicação de Gerência. Neste último componente, inclusive, o Projeto *ServiceUI* é combinado ao Projeto *Surrogate* de forma que o dispositivo relativo à Aplicação de Gerência possa modificar dinamicamente o seu *display* para gerenciar amigavelmente uma vasta gama de dispositivos, que inclui desde dispositivos do tipo *desktop* até aqueles com baixo poder computacional.

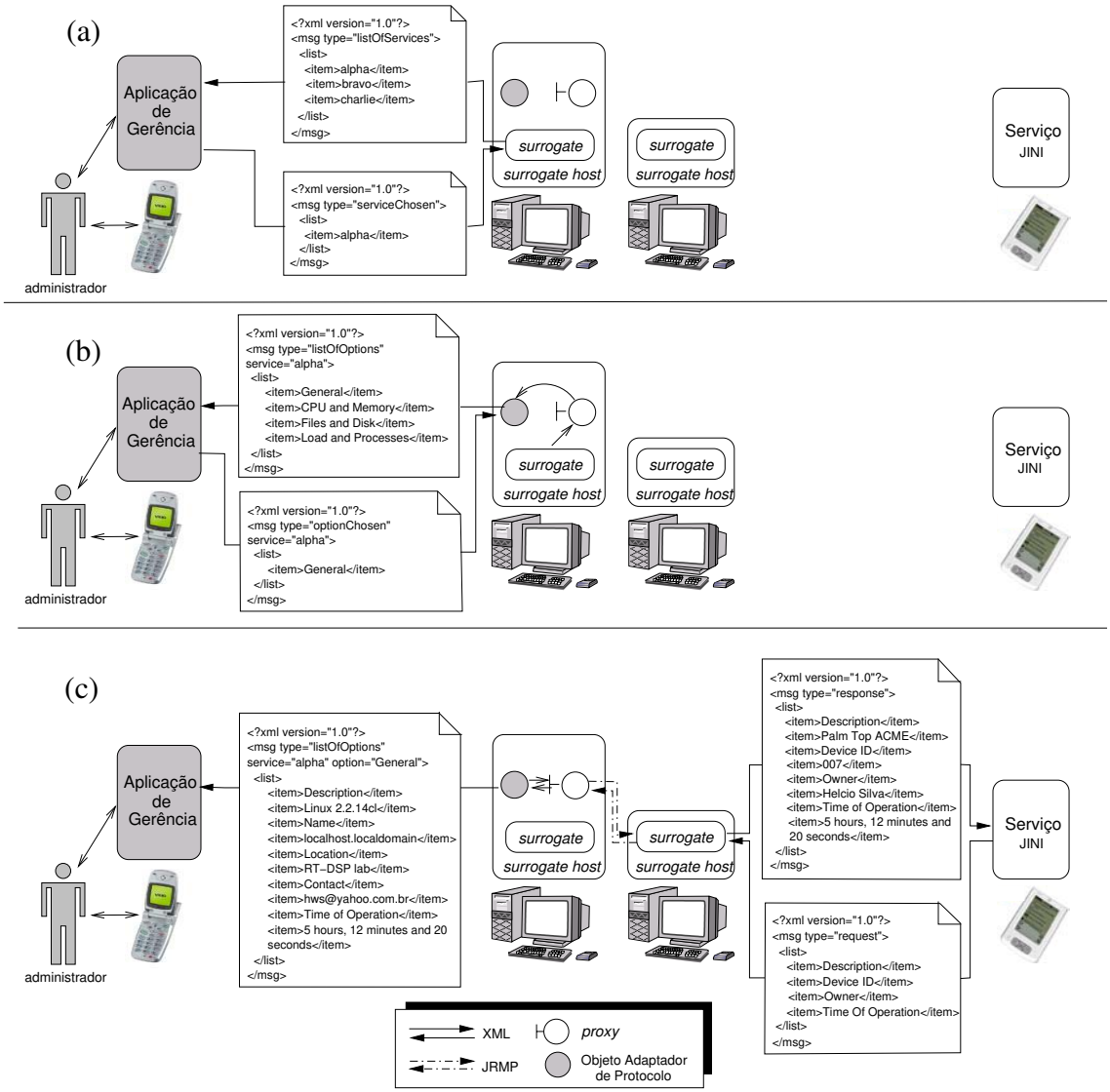

Figura 5.11: Quarto cenário de operação concebido.

Similarmente ao que foi descrito há pouco para o segundo cenário de operação, o dispositivo relativo à Aplicação de Gerência descobre o *surrogate host* e registra junto àquele componente o seu *surrogate*. Esse *surrogate* envia uma mensagem `lisOfServices` para o dispositivo, após descobrir um determinado LUS e obter os *proxies* para os Serviços sob sua responsabilidade. O usuário recebe a lista desses Serviços na tela do dispositivo, e faz a sua escolha. Como resultado, uma mensagem `serviceChosen` é enviada ao *surrogate* informando-lhe acerca do Serviço escolhido (a).

O *surrogate* extrai o Objeto Adaptador de Protocolo a partir do *proxy* para o Serviço escolhido pelo usuário, e aquele Objeto envia uma mensagem `listOfOptions` para o dispositivo. Seu *display* é ajustado novamente, de forma a mostrar a lista de opções disponibilizadas por aquele Serviço. O usuário escolhe uma dessas opções, e uma mensagem `optionChosen` é enviada para o Objeto Adaptador de Protocolo contendo aquela escolha (b).

Após a recepção da mensagem `optionChosen`, o Objeto Adaptador de Protocolo contacta o seu respectivo *proxy*, invocando o método correspondente à opção escolhida pelo usuário. Essa invocação é realizada utilizando o JRMP, sendo endereçada ao *surrogate* implementando a lógica do Serviço correspondente ao *proxy*. Esse *surrogate* contacta o seu dispositivo utilizando o protocolo *request/response* descrito na Subseção 5.3.2. Utilizando esse protocolo, as informações solicitadas são obtidas pelo *surrogate*, que as encaminha para o *proxy*. Essas informações são encapsuladas numa mensagem `listOfData`, e essa mensagem é enviada pelo Objeto Adaptador de Protocolo para o dispositivo correspondente à Aplicação de Gerência. Naquele dispositivo, a mensagem é interpretada e o seu *display* é modificado mais uma vez de forma a exibir essas informações ao usuário (c).

5.5.5 Análise dos resultados

Para analisar os resultados obtidos a partir do emprego dos quatro cenários concebidos para a Baía de Dispositivos, alguns parâmetros são definidos. O primeiro deles é denominado *tempo de setup* (t_s). Este parâmetro corresponde ao intervalo de tempo gasto pela Aplicação de Gerência com os procedimentos necessários até que o *proxy* para um determinado Serviço esteja descarregado e apto a receber invocações. O tempo de *setup* pode ser expresso como a soma de outros dois parâmetros, conforme descrito na Equação a seguir.

$$t_s = t_u + t_d$$

O parâmetro t_u é criado de forma a acomodar nesta análise a Aplicação de Gerência associada a dispositivos com baixo poder computacional (também chamada de "Aplicação de Gerência *surrogate*" na Tabela 5.4). Em particular, o valor deste parâmetro leva em consideração o intervalo de tempo gasto para descobrir um *surrogate host*, registrar com ele um *surrogate* e ter este *surrogate* ativado pelo *surrogate host*. Obviamente, para a chamada "Aplicação de Gerência *non-surrogate*" este parâmetro é nulo.

O parâmetro t_d, por sua vez, diz respeito ao intervalo de tempo necessário para descobrir um LUS, efetuar uma busca junto àquele componente por um determinado Serviço e descarregar o *proxy*

correspondente àquele Serviço. Esses procedimentos são realizados pelos dois tipos de Aplicação de Gerência construídos, a diferença reside apenas no espaço de endereçamento utilizado para fazê-lo.

Os parâmetros t_u e t_d podem ser denominados informalmente de *tempo de upload* (do *surrogate*) e *tempo de download* (do *proxy*), respectivamente. Porém, conforme descrito pelos parágrafos anteriores, essas denominações não são precisas à medida em que não refletem a quantidade de procedimentos envolvidos na obtenção desses parâmetros.

O gráfico de barras ilustrado pela Figura 5.12 mostra os valores de t_s para os cenários concebidos[1]. Conforme se pode observar, esse parâmetro é maior nos cenários que contém a Aplicação de Gerência *surrogate*. Isso é natural, haja vista que naqueles cenários ocorrem os processos de descoberta de *surrogate host*, registro de um *surrogate* e ativação deste *surrogate*. Em outras palavras, naqueles cenários o valor de t_u não é nulo, contribuindo para elevar o valor de t_s. Observa-se inclusive que os valores de t_s para os cenários que incluem a Aplicação de Gerência *surrogate* são aproximadamente iguais, a pequena diferença sendo devida ao tamanho das Classes carregadas em cada um daqueles cenários.

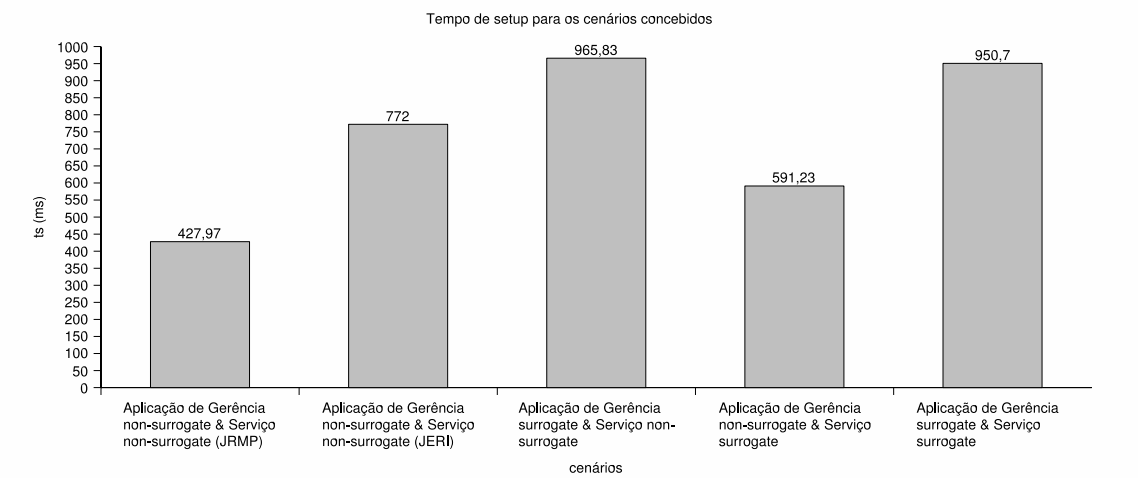

Figura 5.12: Tempos de *setup* para os cenários concebidos.

Outro parâmetro útil para análise dos cenários é o denominado *tempo de invocação* (t_i). Este parâmetro é composto pela soma de duas parcelas, conforme ilustra a Equação a seguir.

$$t_i = t_p + t_r$$

O parâmetro t_p corresponde ao tempo gasto na preparação do *proxy* antes da invocação de métodos. Naturalmente, esse parâmetro é nulo para aplicações que façam uso de versões de JINI anteriores a 2.0, visto que o processo de preparação de *proxies* surgiu somente a partir daquela versão.

[1] os valores mostrados são médias calculadas a partir de conjuntos formados por 100 amostras.

O parâmetro t_r corresponde simplesmente ao *round trip delay* observado na invocação direta de um método junto ao *proxy* para um Serviço.

O gráfico de barras ilustrado pela Figura 5.13 mostra os valores de t_i para os cenários concebidos, utilizando a mesma metodologia descrita para a obtenção dos valores de t_s. Conforme pode-se observar, o maior valor de t_i corresponde ao cenário no qual são empregadas as versões *surrogate* da Aplicação de Gerência e do Serviço construídos. Este valor é quase nove vezes maior que o verificado para o cenário no qual são empregadas as versões *non-surrogate* para esses componentes, em ambos os cenários utilizando-se o mesmo protocolo (JRMP) nas invocações remotas de método.

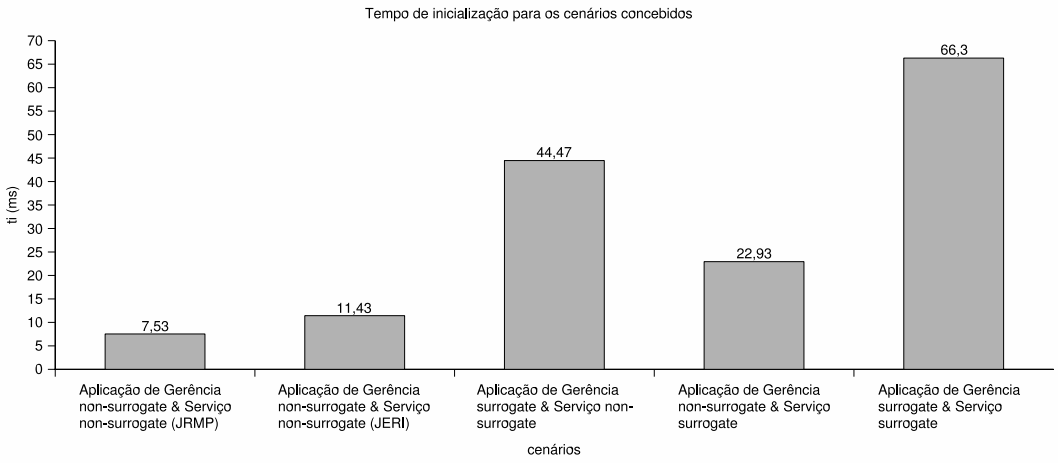

Figura 5.13: Tempos de invocação de método para os cenários concebidos.

A diferença entre esses valores extremos de t_i ilustra bem a diferença de eficiência entre as comunicações baseadas em RMI e aquelas baseadas na troca de mensagens XML. Nesse contexto, outras comparações também podem ser feitas, a saber: na comunicação com a versão *surrogate* do Serviço, a versão *surrogate* da Aplicação de Gerência possui um tempo de invocação quase três vezes maior que o observado para a versão *non-surrogate* desse componente; na comunicação com a versão *non-surrogate* do Serviço, a versão *surrogate* da Aplicação de Gerência possui um tempo de invocação quase seis vezes maior que o observado para a versão *non-surrogate* desse componente.

Uma última observação é feita na comparação dos valores de t_i para a comunicação entre as versões *non-surrogate* da Aplicação de Gerência e do Serviço construídos, quando ambos empregam JRMP e JERI. Neste caso, a aplicação de SSL nas comunicações entre estes componentes utilizando JERI proporcionam um tempo de invocação 52 % maior que aquele verificado quando tais comunicações são feitas utilizando JRMP.

A Tabela 5.5 sumariza os tempos de *setup* e de invocação de método para cada um dos cenários concebidos.

cenário	t_s (ms)	t_i (ms)
Aplicação de Gerência & Serviço *non-surrogates* (JRMP)	427,97	7,53
Aplicação de Gerência & Serviço *non-surrogates* (JERI)	772,00	11,43
Aplicação de Gerência *surrogate* & Serviço *non-surrogate*	965,83	44,47
Aplicação de Gerência *non-surrogate* & Serviço *surrogate*	591,23	22,93
Aplicação de Gerência & Serviço *surrogates*	950,70	66,30

Tabela 5.5: Tempos de *setup* (t_s) e de invocação de método (t_i) para os cenários concebidos.

5.6 Conclusão

Este capítulo abordou uma das provas de conceitos definidas para validação das considerações realizadas no capítulo anterior sobre a utilização de JINI na construção de NMSs. No protótipo desenvolvido, foram construídas duas classes de Aplicação de Gerência e duas classes de Serviços, com vistas a aplicar a quinta daquelas considerações: a incorporação do Projeto *Surrogate* de forma a possibilitar que dispositivos com baixo poder computacional possam fazer parte de um NMS.

Os Serviços JINI construídos para a Baía de Dispositivos foram modelados com o menor grau de granularidade possível, sendo agrupados em Domínios Gerenciados. Objetos de Interface com o Usuário e Objetos Adaptadores de Protocolo foram incorporados aos *proxies* para esses Serviços, de forma a possibilitar a sua gerência amigável tanto por parte de Aplicações de Gerência associadas a dispositivos do tipo *desktop* quanto por parte de versões deste componente associadas a dispositivos com baixo poder computacional. Por fim, os mecanismos de segurança providos a partir da versão 2.0 de JINI foram utilizados nas comunicações entre as Aplicações de Gerência e os Serviços, a exceção do que ocorria quando pelo menos um desses componentes era associado a dispositivos com baixo poder computacional.

Foram definidos quatro cenários de operação, vislumbrando cada uma das interações possíveis entre os componentes desenvolvidos. Em seguida, os resultados da utilização daqueles cenários foram analisados, tendo sido tal análise conduzida por meio de dois parâmetros definidos neste estudo, que são o tempo de *setup* (t_s) e o tempo de invocação de método (t_i). Os resultados desta análise comprovaram que as Aplicações de Gerência associadas a dispositivos com baixo poder computacional possuem um tempo de *setup* bem maior que os seus pares associados a dispositivos do tipo *desktop*. Também foi comprovado que a utilização do Projeto *Surrogate*, a despeito de seu inegável benefício, aumenta o tempo de invocação de métodos junto a um Serviço. Por fim, foi visto que a aplicação de mecanismos de segurança intercambiáveis nas comunicações entre Aplicações de Gerência e Serviços utilizando o JERI tornam estas comunicações mais lentas do que se estivessem utilizando o JRMP, o que também era esperado.

Muito embora a arquitetura apresentada para a Baía de Dispositivos seja centralizada em torno da Aplicação de Gerência, modelos mais descentralizados podem também ser propostos objetivan-

do uma melhor escalabilidade. Isso requer, naturalmente, que uma ou mais Aplicações de Gerência implementem interfaces bem conhecidas, ou seja, ajam também como Serviços JINI. No entanto, haja vista que a arquitetura atual cumpre o seu objetivo, que é validar as considerações gerais elaboradas, essa função não foi implementada na arquitetura atual.

Capítulo 6

Segunda prova de conceitos: desenvolvimento de um PNMS

6.1 Introdução

Conforme abordado no Capítulo 2, um PNMS é composto por uma PMA, um ou vários PDPs e um ou vários PEPs. Adicionalmente, foi comentado naquele capítulo que não há nenhum protocolo estritamente definido para a comunicação entre uma PMA e um PDP.

Por sua vez, o Capítulo 3 mostrou que sistemas distribuídos construídos com base em JINI não requerem a definição estrita de nenhum protocolo, além de levarem vantagem de outras características-chave presentes naquela tecnologia para maximizar a flexibilidade e automação de suas operações, minimizando a possibilidade de intervanção humana na ocorrência de falhas parciais.

A combinação dos fatos relembrados pelos dois parágrafos anteriores é a motivação por trás deste presente capítulo, que descreve a segunda prova de conceitos referente às considerações apresentadas no Capítulo 4 sobre a utilização da tecnologia JINI na construção de NMSs. Para tal, ele é organizado da seguinte forma: na Seção 6.2 é fornecida uma visão geral do PNMS desenvolvido. A Seção 6.3 descreve o tipo de PDP (Serviço JINI) desenvolvido, enquanto a Seção 6.4 descreve a PMA desenvolvida. A Seção 6.5 provê um exemplo de operação do PNMS. Já a Seção 6.6 descreve uma proposta alternativa de PNMS, guardando semelhanças e diferenças em relação ao sistema descrito nas seções anteriores. A Seção 6.7 finaliza o Capítulo, efetuando as considerações finais acerca desses sistemas.

6.2 Visão geral do PNMS

A Figura 6.1 ilustra uma visão geral do protótipo de PNMS desenvolvido. Nele, foram seguidas as considerações realizadas na Seção 4.2 e o PDP foi modelado como um Serviço JINI. Essa medida de projeto garante a flexibilidade na adoção do protocolo utilizado nas comunicações entre a PMA e o PDP, assim como contribui para a auto-resiliência do PNMS como um todo.

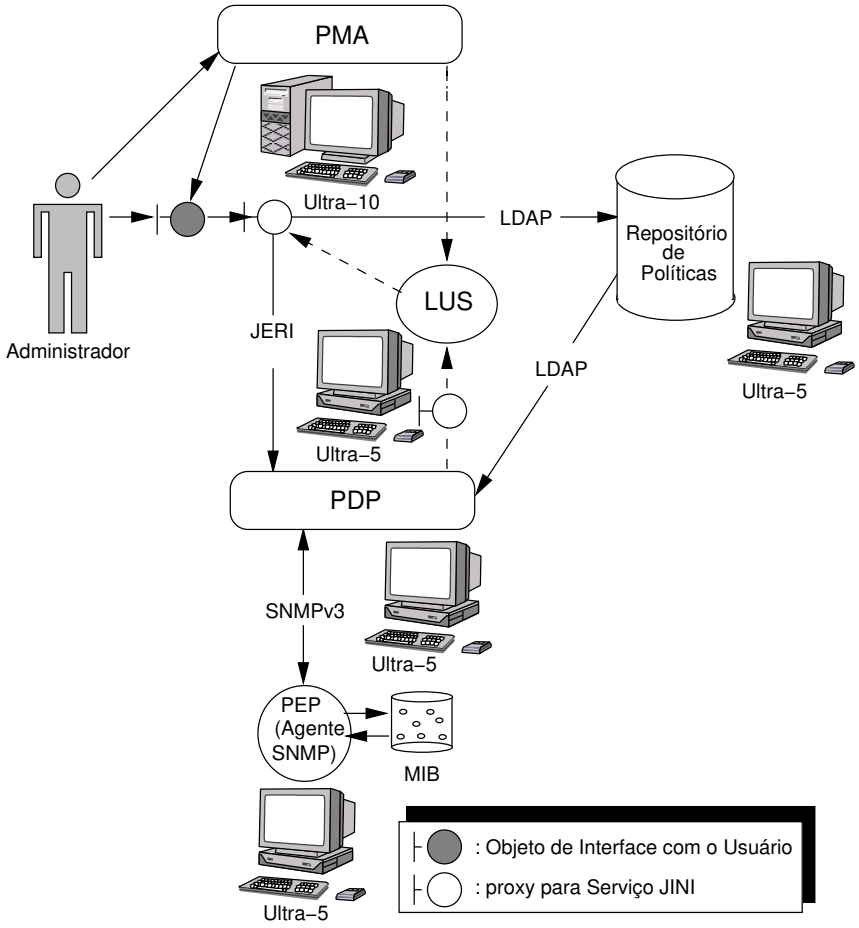

Figura 6.1: Visão geral do PNMS desenvolvido.

Também é notável na Figura 6.1 a incorporação de um Objeto de Interface com o Usuário ao *proxy* para o PDP, a partir do qual o usuário da PMA pode realizar a gerência daquele componente. Essa é uma aplicação da consideração feita inicialmente na Seção 4.4.

Embora o protótipo implementado contenha apenas um PDP, mais componentes deste tipo podem aderir ao seu Domínio Gerenciado correspondente, usufruindo dos benefícios comentados na Seção 4.3.

As interações entre a PMA e o PDP são realizadas utilizando-se JERI. Seguindo-se a consideração feita inicialmente na Seção 4.5, são aplicadas restrições relativas à obrigatoriedade de autenticação

e privacidade nas comunicações entre a PMA e o PDP, o que se traduziu na utilização do SSL nas comunicações entre essas aplicações. Ademais, são utilizadas URLs HTTPMD de forma a garantir a integridade do código descarregado dinamicamente através da infra-estrutura de rede.

No protótipo desenvolvido, o PEP foi representado pelo *daemon* `snmpd` incluso na versão 5.2.1 do NET-SNMP. Nesse PNMS, as interações entre o PDP e o PEP ocorrem através da troca de mensagens SNMPv3, o PDP configurando gatilhos na MIB de Evento suportada pelo `snmpd` e eventualmente recebendo PDUs *Trap* quando eventos são disparados.

O Repositório de Políticas foi representado pelo *daemon* `slapd` incluso na versão 2.1.25 do OpenLDAP, uma implementação de código aberto do LDAP [The OpenLDAP Foundation, 2005]. Assim como ocorre para o `snmpd`, o `slapd` é configurado a partir de um arquivo.

As implementações do PDP, PEP e do Repositório de Políticas são executadas a partir de estações Sun Ultra-5, equipadas com processadores operando a 270 MHz e possuindo 128 MBytes de RAM. Já a PMA é executada a partir de uma estação Ultra 10 equipada com um processador operando a 300 MHz e possuindo 256 MBytes de RAM. Essas plataformas são equipadas com a versão 9 do Sistema Operacional Solaris.

A PMA e o PDP foram desenvolvidos utilizando a versão 1.4.2_04 da J2SE e a versão 2.0_002 do JTSK. Na construção desses componentes, fez-se uso da API JAAS de forma que apenas usuários devidamente autenticados pudessem ter acesso as suas respectivas configurações.

6.3 O PDP desenvolvido

O PDP foi desenvolvido utilizando as versões 1.4.2_04 da J2SE e 2.0_002 do JTSK, respectivamente. Na construção desse componente, fez-se uso da API JAAS de forma que apenas um usuário devidamente autenticado pudesse ter acesso a sua configuração.

A Figura 6.2 ilustra a GUI para o PDP desenvolvido, particularmente num momento no qual ele conclui sua adesão a um determinado Domínio Gerenciado. Conforme pode ser observado, para a sua inicialização é necessário configurar-se parâmetros alusivos ao PEP (Agente SNMPv3) com o qual irá comunicar-se, ao Repositório de Políticas (Servidor LDAP) a partir do qual coletará Regras de Política colocadas previamente pela PMA e, finalmente, ao LUS responsável pelo Domínio Gerenciado ao qual irá pertencer.

Para a comunicação com o Agente SNMPv3, são necessários o endereço IP do Agente, o *User Name* a ser utilizado nas Mensagens trocadas, uma senha de autenticação e o *Engine ID* a ser utilizado pelo PDP para a recepção de eventuais notificações acerca de gatilhos disparados no Agente.

Para a comunicação com o Servidor LDAP, são necessários o endereço IP do Servidor, uma senha de autenticação e um DN (*Distinguished Name*) a partir do qual as Regras de Política serão buscadas no Servidor.

Para a comunicação como LUS, são necessários o nome administrativo do PDP e o nome do Domínio Gerenciado ao qual deseja-se que ele pertença. São também disponibilizadas as opções de escolha para o método a ser utilizado para encontrar-se o LUS (*multicast* ou *unicast*) e o método de renovação do *lease* junto àquele Serviço (renovação automática ou com parâmetros definidos pelo

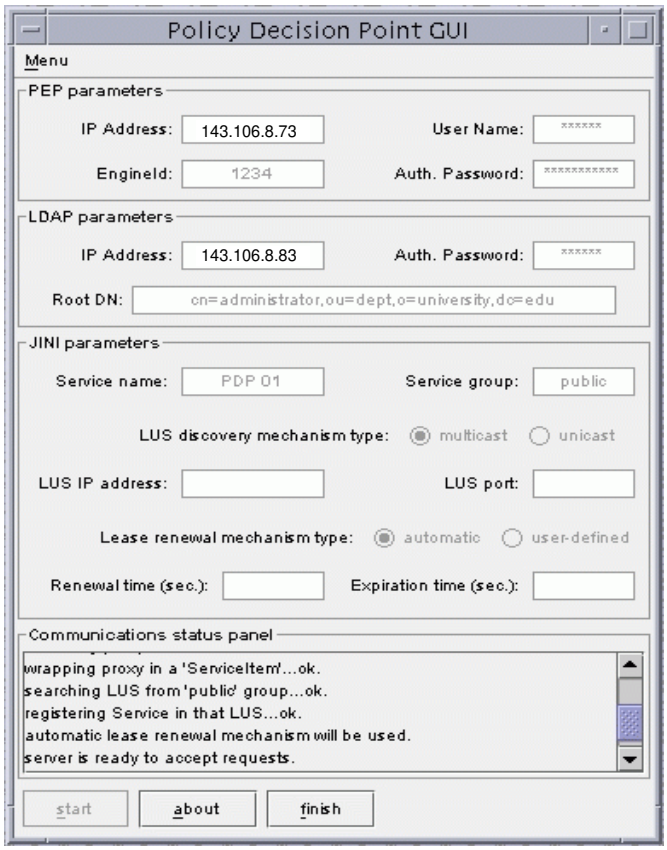

Figura 6.2: GUI para o PDP desenvolvido.

usuário).

O PDP, como qualquer Serviço JINI, implementa uma determinada interface. Em particular, a interface implementada no protótipo é ilustrada a seguir:

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface PDPService extends Remote {
  public void activate(String ruleName)
    throws RemoteException;
  public void deactivate(String ruleName)
    throws RemoteException;
  public String[] list() throws RemoteException;
}
```

Conforme observado a partir desta interface, o PDP provê suporte para a ativação ou a desativação de uma determinada Regra de Política cujo nome administrativo é passado como parâmetro, bem como para a obtenção de uma listagem das Regras de Política nele atualmente ativas (isto é, Regras para as quais existem entradas nas tabelas da MIB de Evento no Agente subordinado ao PDP).

Para a comunicação com o Repositório de Políticas, é utilizada a API JLDAP, provida pela Novell e disponibilizada gratuitamente. Para a comunicação com o Agente SNMP (PEP), é utilizada a API SNMPv3 provida pela AdventNet (mais especificamente, a versão 3.3, outrora disponível gratuitamente para descarregamento a partir da rede).

6.4 A PMA desenvolvida

A semelhança do que foi comentado com relação ao PDP, a PMA também foi desenvolvida utilizando as versões 1.4.2_04 e 2.0_002 da J2SE e do JTSK, respectivamente. Além disso, também foi utilizada a API JAAS para a garantia de que apenas um usuário devidamente autenticado pudesse ter acesso a sua configuração.

A Figura 6.3 ilustra a GUI para a PMA desenvolvida, particularmente num momento no qual ela conclui sua inicialização. Para que isso ocorra, é necessário o fornecimento do nome de um Domínio Gerenciado e a escolha do método a ser utilizado para efetuar-se a busca pelo LUS responsável por esse Domínio.

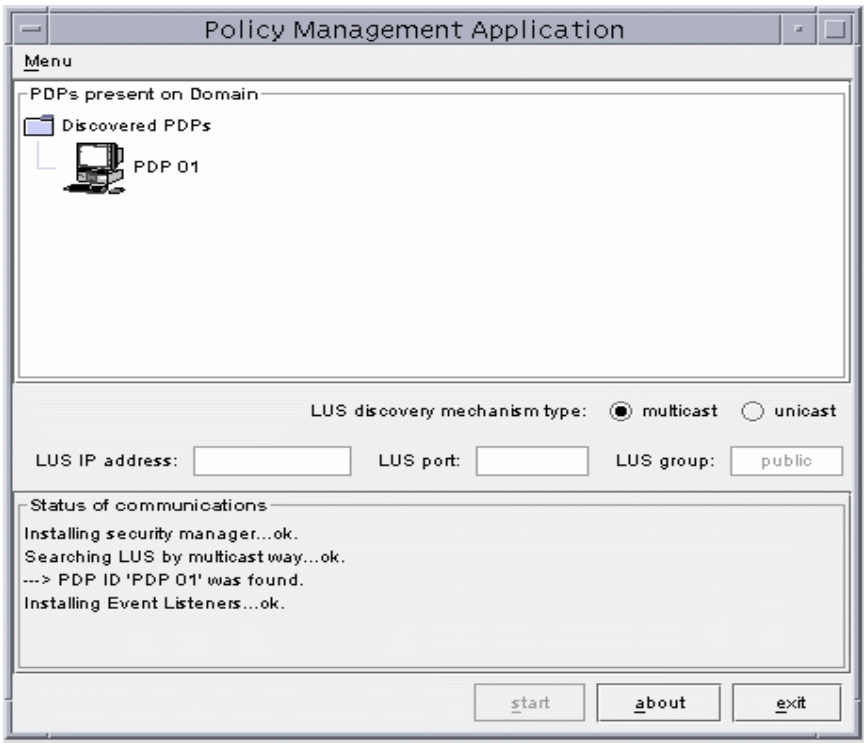

Figura 6.3: GUI para a PMA desenvolvida.

A inicialização da PMA compreende a busca do LUS responsável pelo Domínio Gerenciado

fornecido, a busca imediata de PDPs que já pertencem a este Domínio e a instalação de Ouvidores de Evento junto ao LUS que denunciam a entrada ou a saída de PDPs do Domínio.

Na GUI mostrada na Figura 6.3, o Domínio buscado já contém um PDP associado. Este PDP é representado na tela por um ícone localizado numa estrutura em árvore na GUI. Essa estrutura é destinada a abrigar todos os os PDPs presentes naquele Domínio.

6.5 Exemplo de Operação

As operações de gerência neste PNMS compreendem fundamentalmente a invocação remota dos métodos pertencentes à interface disponibilizada pelo PDP, a qual foi descrita na Seção 6.3. Para que isso ocorra, o usuário da PMA deve selecionar um dos PDPs presentes no Domínio e obter dinamicamente a interface (gráfica) para gerenciá-lo.

A Figura 6.4 ilustra a GUI que é descarregada dinamicamente para a gerência do PDP, particularmente num momento no qual o método `activate` pertencente à interface do PDP é invocado.

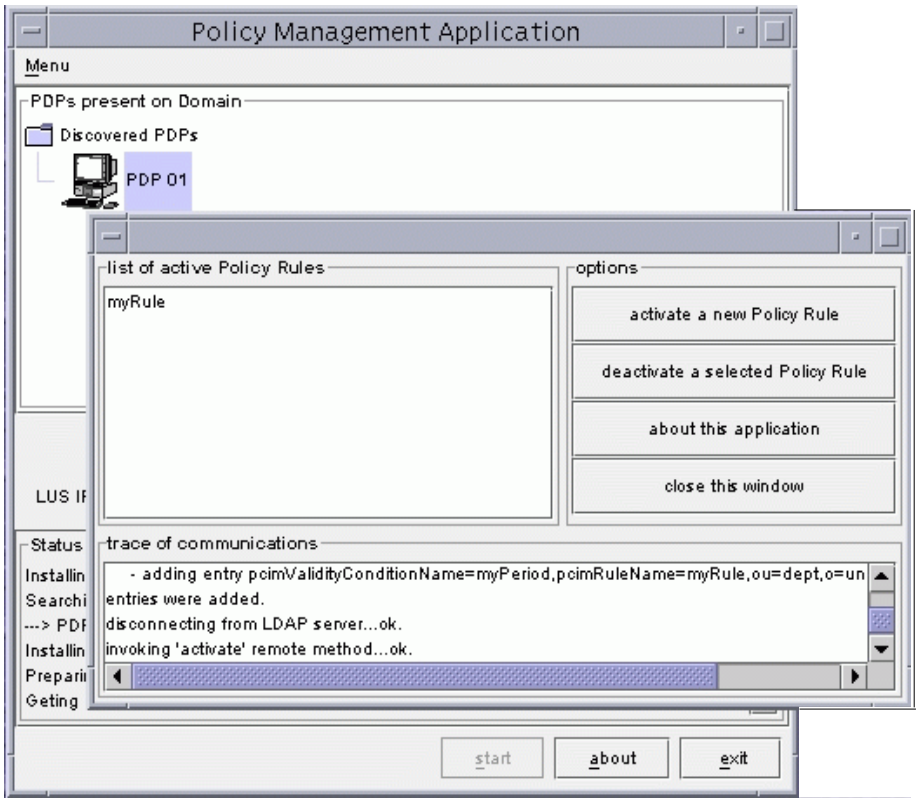

Figura 6.4: GUI descarregada dinamicamente para a gerência do PDP.

No PNMS desenvolvido, uma Regra de Política é inicialmente expressa utilizando-se a XML,

seguindo-se a DTD (*Document Type Definition*) descrita no Apêndice A. Essa Regra de Política pode ser criada a partir de um editor de texto qualquer, sendo armazenada num arquivo no disco da máquina na qual a PMA reside.

Como exemplo, considere-se a seguinte Regra de Política:

> *Durante o período que vai das 08:00:00 do dia 15 de abril de 2005 até às 18:00:00 do dia 15 de maio de 2005, checar o valor do MO* `ifInOctets` *para cada interface do Sistema Gerenciado a cada 1 (um) minuto. Caso a diferença entre dois valores consecutivos ultrapasse 30, enviar uma notificação para o 'host' 143.106.8.78 contendo o valor do MO* `sysUpTime` *do Sistema Gerenciado.*

Esta Regra de Política é assim inicialmente expressa:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE policy SYSTEM "policy.dtd">
<!-- definition of a Policy example -->
<policy name="myRule" status="enabled"
   conditionListType="dnf" priority="1">
  <!-- definition of a Policy condition -->
  <condition name="myCondition" groupNumber="1"
    negated="false">
    <objectTarget wildcarding="true">
      <objectTargetID>ifInOctets</objectTargetID>
      <objectTargetType>Counter</objectTargetType>
    </objectTarget>
    <test frequency="60" delta="true">
      <threshold type="rising">
        <thresholdValue>30</thresholdValue>
      </threshold>
    </test>
  </condition>
  <!-- definition of a Policy action -->
  <action name="myAction" order="1">
    <notification destination="143.106.8.78">
      <objectNotif wildcarding="false">
        <objectNotifID>sysUpTime</objectNotifID>
        <objectNotifType>TimeTicks</objectNotifType>
      </objectNotif>
    </notification>
  </action>
  <!-- definition of a time Policy condition -->
  <timePeriod name="myPeriod">
    <initial>
      <year>2005</year>
      <month>04</month>
      <day>15</day>
      <hours>08</hours>
      <minutes>00</minutes>
```

```
      <seconds>00</seconds>
    </initial>
    <final>
      <year>2005</year>
      <month>05</month>
      <day>15</day>
      <hours>18</hours>
      <minutes>00</minutes>
      <seconds>00</seconds>
    </final>
  </timePeriod>
</policy>
```

Para ativar uma Regra de Política, procede-se ao carregamento do seu respectivo arquivo XML e a sua conversão em Objetos PCLS.

A Regra de Política mostrada antes, após a conversão, é mostrada a seguir no formato LDIF (*LDAP Data Interchange Format*)[1].

```
dn: pcimRuleName=myRule,ou=dept,o=university,dc=edu
objectclass: top
objectclass: dlm1ManagedElement
objectclass: pcimPolicy
objectclass: pcimRule
objectclass: pcimRuleInstance
pcimRuleName: myRule
pcimRuleEnabled: 1
pcimRuleConditionListType: 1
pcimRulePriority: 1
pcimRuleConditionList: pcimConditionName=myCondition,
pcimRuleName=myRule,ou=dept,o=university,dc=edu
pcimRuleValidityPeriodList: pcimValidityConditionName=
myPeriod,pcimRuleName=myRule,ou=dept,o=university,dc=edu
pcimRuleActionList: pcimActionName=myAction,pcimRuleName=
myRule,ou=dept,o=university,dc=edu

dn: pcimConditionName=myCondition,pcimRuleName=myRule,
ou=dept,o=university,dc=edu
objectclass: top
objectclass: dlm1ManagedElement
objectclass: pcimPolicy
objectclass: pcimRuleConditionAssociation
objectclass: pcimConditionAuxClass
objectclass: myConditionAuxClass
pcimConditionName: myCondition
```

[1]LDIF é um formato para definição de entradas de um diretório no formato texto, sendo especificiado em [Good, 2000]. A maioria dos serviços de diretório permite a importação de entradas a partir de um arquivo no formato LDIF.

```
pcimConditionGroupNumber: 1
pcimConditionNegated: FALSE
objectID: ifInOctets
objectType: Counter
wildcarding: TRUE
testType: 3
samplingType: 2
samplingFreq: 60
targetValue: 30
thresholdType: 1

dn: pcimValidityConditionName=myPeriod,pcimRuleName=myRule,
ou=dept,o=university,dc=edu
objectclass: top
objectclass: dlm1ManagedElement
objectclass: pcimPolicy
objectclass: pcimRuleValidityAssociation
objectclass: pcimTPCAuxClass
pcimValidityConditionName: myPeriod
pcimTPCTime: 20050415T080000/20050515T180000

dn: pcimActionName=myAction,pcimRuleName=myRule,ou=dept,
o=university,dc=edu
objectclass: top
objectclass: dlm1ManagedElement
objectclass: pcimPolicy
objectclass: pcimRuleActionAssociation
objectclass: pcimActionAuxClass
objectclass: myActionAuxClass
pcimActionOrder: 1
pcimActionName: myAction
eventType: 2
destination: 143.106.8.78
objectNotifID: sysUpTime
objectNotifType: TimeTicks
```

No protótipo desenvolvido, o *schema* PCLS foi incorporado ao conjunto de *schemas* suportados pelo `slapd`. Além disso, um novo *schema* foi desenvolvido para dar suporte ao mapeamento entre Objetos aderentes ao PCLS e MOs aderentes à SMI, de forma que o PDP pudesse estabelecer qualquer um dos tipos de testes previstos na especificação da MIB de Evento.

Conforme ilustrado pela Figura 6.5, nesse novo *schema* são definidas as Classes Auxiliares `myCondition` e `myAction`. O Apêndice B contém a especificação detalhada dessas Classes e de seus respectivos atributos.

Uma vez carregada e convertida em Objetos PCLS, a Regra de Política é escrita num Repositório de Políticas. Em seguida, uma chamada remota ao método `activate` é realizada automaticamente junto ao *proxy* para o PDP.

Quando o PDP recebe essa invocação, ele procede imediatamente a busca da Regra de Política (cujo nome é passado como parâmetro do método) junto ao Repositório de Políticas.

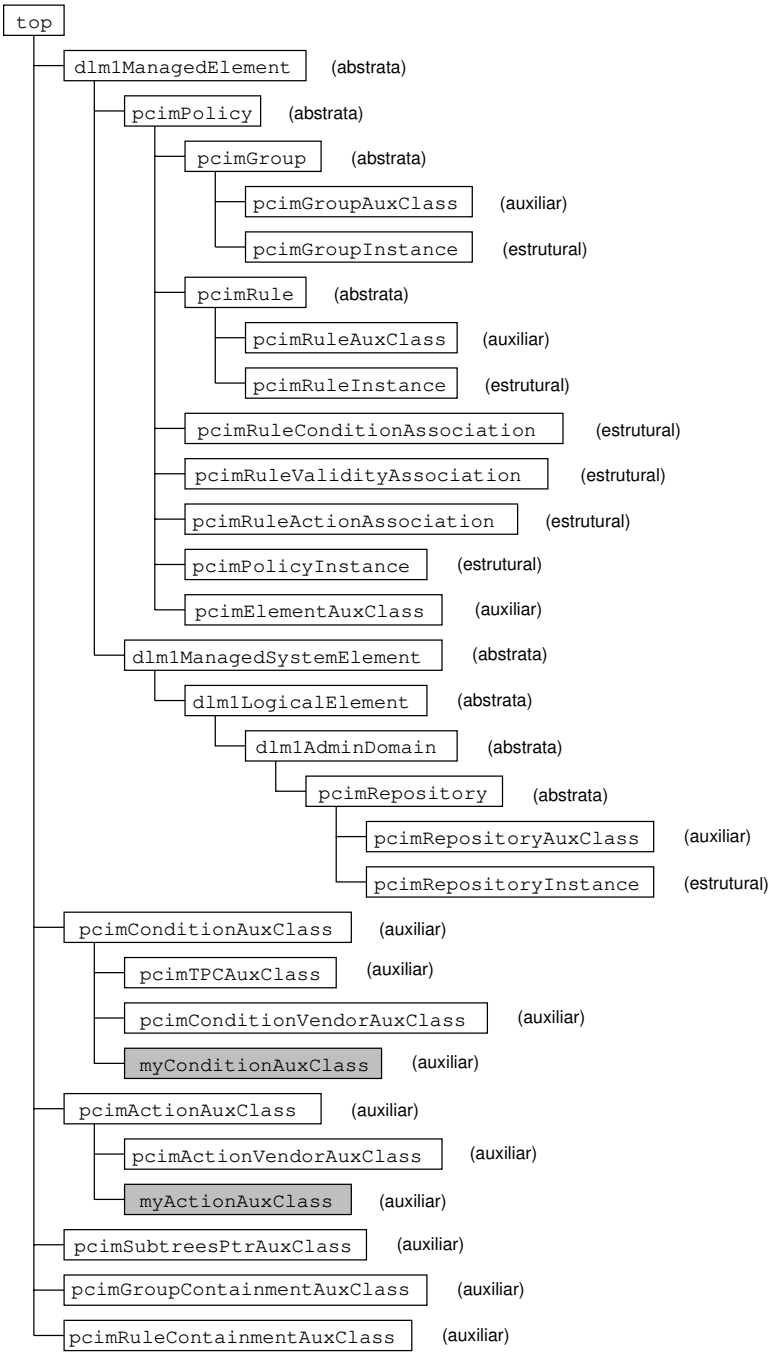

Figura 6.5: Extensões ao PCLS desenvolvidas para o protótipo.

Conforme comentado na Seção 6.2, este PNMS faz uso do protocolo SNMP e da MIB de Evento para o reforço da Regra de Política junto a um PEP. Por isso, na implementação do método `activate`

no PDP, os Objetos PCLS coletados no Repositório de Políticas são convertidos em MOs aderentes à SMI e em informações alusivas aos períodos segundo os quais a Regra deve ser reforçada.

A ativação da Regra de Política tomada como exemplo corresponde inicialmente à criação de um gatilho, adicionando uma nova entrada na tabela `mteTriggerTable`. Em seguida, é criada uma nova entrada na tabela `mteEventTable` para o tipo de ação a ser realizada quando o gatilho é disparado que, nesse caso, é o envio de uma PDU *Trap*. Assim, é adicionada uma nova entrada na tabela `mteEventNotificationTable`, que passa a conter uma lista de MOs correspondente à entrada na tabela `mteEventTable`.

Uma vez definidos os parâmetros gerais do gatilho e configuradas as ações a serem levadas a cabo na ocorrência de um evento pelo disparo daquele gatilho, são definidos então seus parâmetros específicos. Dependendo do tipo de teste a ser realizado com o MO monitorado, o gatilho recebe a denominação de gatilho de existência, de valor-limite ou gatilho *booleano*. No caso da Regra de Política tomada como exemplo, trata-se de um gatilho de valor-limite. Assim, a próxima (e última) tabela que recebe uma nova entrada é a tabela `mteTriggerThresholdTable`.

A desativação de uma determinada Regra de Política, iniciada a partir da invocação do método `deactivate`, é implementada com a eliminação de suas respectivas entradas nas tabelas da MIB de Evento. Já a listagem de Regras de Política atualmente ativas no PDP, iniciada a partir da invocação do método `list`, é implementada a partir da consulta aos registros internos daquela aplicação.

6.6 Projeto alternativo

A Figura 6.6 descreve um projeto alternativo para o PNMS desenvolvido. Conforme pode-se observar, a principal diferença entre esse novo projeto e aquele comentado ao longo deste capítulo está na definição do PEP, que agora é representado não por um Agente SNMPv3, mas por um Serviço JINI.

Esse novo PEP é estruturado de forma a possuir uma funcionalidade análoga àquela presente no Agente SNMPv3 no que diz respeito à utilização da MIB de Evento. Isso pode ser inferido a partir da interface implementada pelo Serviço, ilustrado a seguir.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import net.jini.core.transaction.Transaction;

public interface PEPService extends Remote {
    // métodos destinados à criação de gatilhos e configuração de
    // seus parâmetros gerais.
    public void createTrigger(String owner, String name, Transaction txn)
throws RemoteException;
    public void setValueID(String oid, Transaction txn)
throws RemoteException;
    public void setValueIDWildcard(boolean option, Transaction txn)
throws RemoteException;
    public void setSampleFrequency(long frequency, Transaction txn)
```

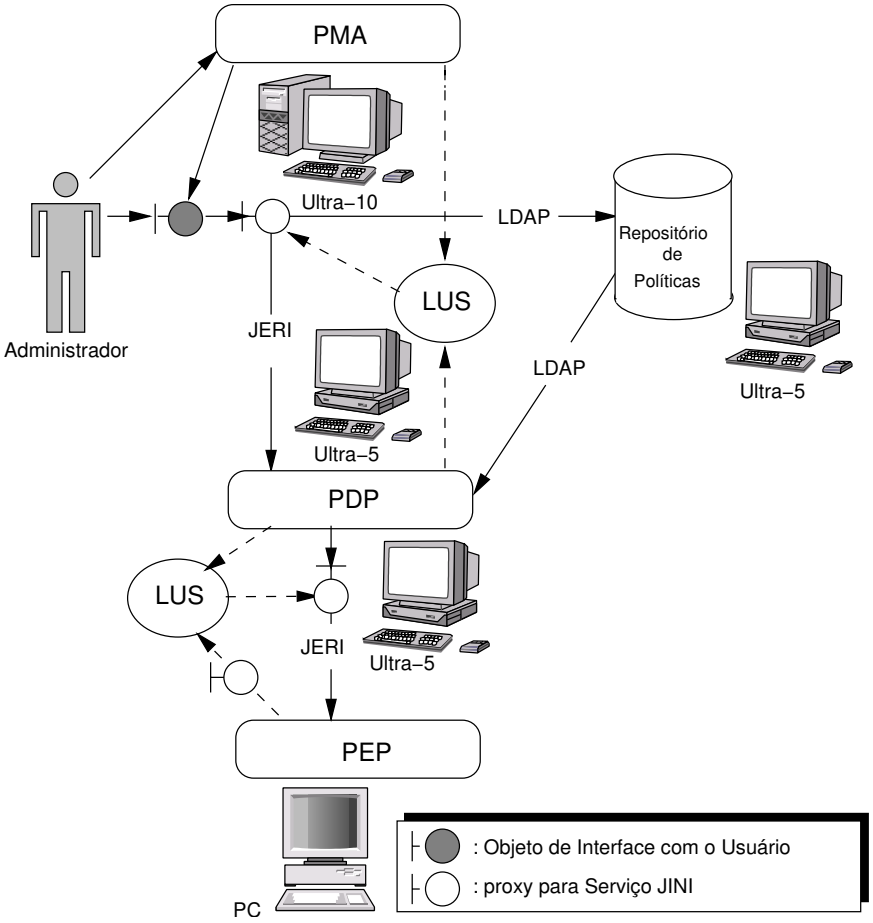

Figura 6.6: Projeto alternativo para o PNMS desenvolvido.

```
throws RemoteException;
    public void setSampleType(int type, Transaction txn)
throws RemoteException;
    public void setTestType(int type, Transaction txn)
throws RemoteException;
    // método destinados à configuração de gatilhos de existência.
    public void setExistenceType(int type, Transaction txn)
throws RemoteException;
    // métodos destinados à configuração de gatilhos do tipo
    // 'boolean'.
    public void setBooleanComparisonType(int type, Transaction txn)
throws RemoteException;
    public void setBooleanComparisonValue(long value, Transaction txn)
throws RemoteException;
    // métodos destinados configuração de gatilhos do tipo
    // valor-limite.
    public void setThresholdType(int type, Transaction txn)
```

```
throws RemoteException;
    public void setThresholdValue(int value, Transaction txn)
throws RemoteException;
    // métodos destinados à criação e configuração de eventos.
    public void createEvent(String name, Transaction txn)
        throws RemoteException;
    public void setAction(int type, Transaction txn)
throws RemoteException;
    public void setObjects(String[] oids, Transaction txn)
        throws RemoteException;
    public void setValues(String[] values, Transaction txn)
        throws RemoteException;
}
```

A Figura 6.7 ilustra um exemplo do reforço de uma Regra de Política nesse novo PEP. A semelhança com o PEP representado pelo Agente SNMP, esse procedimento consiste inicialmente na criação de um gatilho, nesse caso através do método `createTrigger`.

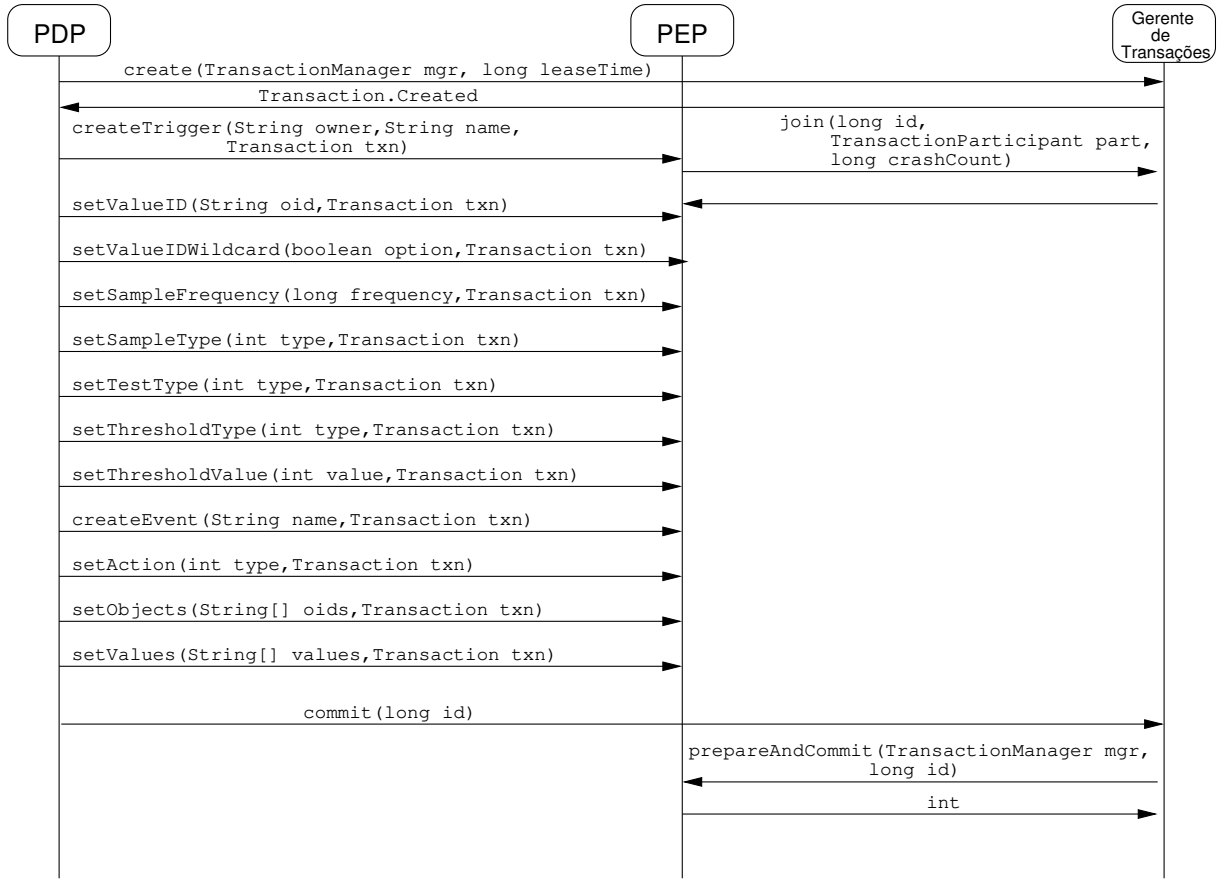

Figura 6.7: Exemplo de reforço de uma Regra de Política no PEP alternativo.

Em seguida, são configurados alguns parâmetros gerais associados a este gatilho, utilizando-se para isso os métodos `setValueID`, `setValueIDWildcard`, `setSampleFrequency`, `setSampleType` e `setTestType`. Esses métodos definem, respectivamente, a identificação de um parâmetro a ser monitorado, a utilização (ou não) de *wildcarding* nessa identificação, a freqüência com que o parâmetro será monitorado (em segundos), o tipo de valor de interesse obtido a partir desse parâmetro (valores absolutos ou a diferença entre dois daqueles valores consecutivos) e, finalmente, o tipo de teste a ser realizado com os valores obtidos (que pode ser um teste de existência, de valor-limite ou um teste do tipo *boolean*).

A presença dos métodos `setThresholdType` e `setThresholdValue` na Figura 6.7 denota a configuração de um gatilho do tipo valor-limite (conforme descrito no parágrafo anterior, essa classificação é estabelecida anteriormente através do método `setTestType`). Esses métodos definem, respectivamente, qual será o tipo de valor-limite considerado para o parâmetro a ser monitorado (valor-limite superior ou inferior) e qual será o seu valor correspondente.

O evento correspondente àquele gatilho é criado a partir do método `createEvent`, e o tipo de ação na qual ele consiste é definido a partir do método `setAction`. Na Figura 6.7, a ação realizada após o disparo do gatilho consiste na configuração local de parâmetros. Os métodos `setObjects` e `setValues` definem, respectivamente, quais são estes parâmetros e quais serão os seus valores correspondentes.

Conforme pode ser observado na Figura 6.7, os mecanismos transacionais descritos na Seção 4.6 são utilizados na invocação dos métodos junto ao PEP. Essa medida tem por finalidade garantir que todas aquelas invocações sejam realizadas em seqüência e com sucesso, fazendo com que o PEP atinja um estado consistente.

Para tal, uma transação é criada pelo PDP junto ao Gerente de Transações (`mahalo`), e um Objeto implementando a interface `net.jini.core.transaction.Transaction` foi passado como argumento em todos os métodos citados anteriormente. Durante a invocação do primeiro deles (`createTrigger`), o PEP aderiu à transação mediante invocação do método `join` junto ao Gerente de Transações.

Nos demais métodos, o Objeto `net.jini.core.transaction.Transaction`, que em última instância contém o identificador da transação, serviu para que as estruturas internas do PEP fossem devidamente preparadas para:

1. sinalizar eventuais alterações na seqüência predeterminada de invocações, nesse caso abortando a transação;

2. efetivar (ou abortar) a transação em decorrência de solicitações provenientes, em última instância, do PDP.

Na Figura 6.7, após a invocação de todos os métodos relativos ao reforço da Regra de Política, o PDP solicita ao Gerente de Transações a efetivação da transação. Ele assim o faz utilizando o método `prepareAndCommit`, pertencente à interface `net.jini.core.transaction.server.TransactionParticipant`. Essa interface foi implementada pelo PEP de forma que o mesmo pudesse agir na condição de Participante da transação.

O projeto desse novo PEP repercute diretamente no projeto do PDP, que sofre modificações. Mais especificamente, nesse novo PNMS os Objetos PCLS coletados são convertidos pelo PDP não em MOs aderentes à SMI, mas sim em argumentos a serem postos em métodos invocados junto ao PEP.

Um ponto notável na Figura 6.6 é a presença de dois LUSs. De fato, o LUS situado no hemisfério inferior daquela figura destina-se primordialmente ao registro de *proxies* para PEPs subordinados ao PDP. Já o LUS localizado em seu hemisfério superior é destinado ao registro de *proxies* para os (potencialmente vários) PDPs sob coordenação da PMA.

Com relação aos dois parágrafos anteriores, uma observação deve ser feita. Haja vista que, segundo a filosofia seguida pelo PNMS descrito nas seções anteriores a esta, o reforço de uma Regra de Política permite também que mensagens assíncronas sejam enviadas pelo PEP ao PDP, tem-se para esse novo projeto duas importante implicações.

A primeira dessas implicações atinge a interface implementada pelo PDP, que deve ser ligeiramente modificada para incluir um novo método que deve ser invocado pelo PEP quando um gatilho é disparado naquele componente. A assinatura deste novo método é mostrada a seguir.

```
public void sendNotification(String[] oids, String[] values)
    throws RemoteException;
```

Os argumentos presentes nesse novo método identificam parâmetros presentes no PEP e os seus respectivos valores no momento em que o gatilho configurado é disparado. A identificação prévia desses parâmetros deve ser feita através de métodos apropriados, que naturalmente constam na especificação da interface.

A segunda implicação, decorrente da primeira, é que o PDP deve registrar-se também junto ao LUS no qual registram-se seus PEPs subordinados, de forma que esses últimos tenham acesso ao seu *proxy*. Uma proposta alternativa a esta seria os referidos PEPs terem acesso ao LUS localizado no hemisfério superior da Figura 6.6. Essa proposta, no entanto, traria sérios problemas de escalabilidade em cenários reais de operação na qual existem vários PEPs subordinados a um mesmo PDP e vários PDPs subordinados a uma mesma PMA.

6.7 Conclusão

Esse capítulo apresentou mais uma prova de conceitos das considerações realizadas no Capítulo 4. Em particular, essa prova de conceitos consistiu no desenvolvimento de um NMS que apresenta a combinação de iniciativas surgidas paralelamente no âmbito da IETF para flexibilizar e descentralizar a gerência de redes.

Foram combinadas uma das iniciativas do *DISMAN WG* (a MIB de Evento) e a iniciativa do *Policy Framework WG* (o paradigma de Gerência Baseada em Políticas) para a construção de um PNMS cujo reforço de Políticas junto ao PEP é viabilizada segundo a monitoração de MOs e a definição de eventos que ocorrem quando gatilhos são disparados com base em testes realizados naqueles MOs.

Uma técnica inovadora empregada na construção do PNMS foi a utilização da tecnologia JINI nas interações entre PMA e PDP. O efeito da aplicação dessa técnica é a concepção de um sistema flexível e auto-resiliente, no qual a PMA mantém uma visão consistente sobre seus PDPs subordinados e é capaz de comunicar-se com esses componentes utilizando diversos tipos de Interface com o Usuário, empregando virtualmente qualquer protocolo.

Adicionalmente, esse capítulo descreveu o projeto de um novo PNMS que, embora totalmente baseado em JINI, ainda preserva a semântica do reforço de Regras de Política adotada no PNMS original. Nesse novo projeto, o PEP foi representado por um Serviço JINI, recebendo invocações remotas do PDP, que age agora também como um Cliente JINI, além de prover Serviços à PMA.

Ao final do tempo decorrido entre a construção do PNMS original e a concepção desse novo projeto, a infra-estrutura computacional destinada a realização de testes não se encontrava mais disponível. Isso fez com que a implementação desenvolvida para o novo PEP não pudesse comunicar-se com os demais componentes, que, no caso do PDP, também sofreria modificações.

Capítulo 7

Comparações entre JINI e outras tecnologias para a construção de NMSs

7.1 Introdução

Comparações entre JINI e outras tecnologias, ou até mesmo produtos, não constituem-se em um assunto novo. De fato, comparações arquiteturais entre JINI e outras tecnologias são realizadas em [Olstad et al., 1999] e [Raza et al., 2000], por exemplo, ambos apontando JINI como escolha preferencial para a construção de aplicações distribuídas.

Em [Olstad et al., 1999], JINI é comparada com produtos como *Millennium* e *Universal Plug and Play* (desenvolvidas pela Microsoft), *Inferno* (desenvolvida pela Lucent) e *T-Spaces* (desenvolvida pela IBM). Esta comparação é realizada levando-se em conta aspectos como serviços, configuração, escalabilidade e portabilidade, e embora o aspecto de custo de aquisição não tenha sido levado em conta, JINI foi apontado como a tecnologia mais promissora para a construção de sistemas distribuídos.

Em [Raza et al., 2000], JINI é comparada com tecnologias como CORBA e Agentes Móveis para provisionamento de configuração de serviços de rede, também tendo sido apontada como a escolha mais adequada em comparação com aquelas tecnologias.

Neste capítulo, JINI é comparada com SNMP, CORBA e Web Services, levando-se em conta a sua aplicabilidade específica na área de gerência de redes. Para tal, a próxima seção descreve algumas limitações observadas no modelo de gerência tradicional baseado no SNMP. Em seguida, a Seção 7.3 descreve algumas iniciativas relacionadas ao emprego de CORBA e Web Services na construção de NMSs. A Seção 7.4 realiza a comparação entre JINI e as demais tecnologias, levando-se em consideração aspectos de modelagem de Serviços, utilização de infra-estrutura existente, possibilidade de emprego de Interfaces com o Usuário flexíveis e aplicabilidade em dispositivos

com baixo poder computacional. Por fim, a Seção 7.5 efetua as considerações finais acerca dessa comparação, encerrando o capítulo.

7.2 Limitações do modelo tradicional

Conforme apontado em [Schönwälder et al., 2003], o modelo tradicional baseado no binômio SNMP/SMI foi elaborado para minimizar o número e complexidade das atividades de gerência realizadas pelos Gerentes e Agentes, além de ser extensível para acomodar funcionalidades adicionais e/ou imprevistas. O modelo foi elaborado também para desempenhar seu papel quando a rede gerenciada não está completamente operacional, para tal sendo associado a um protocolo não-orientado-a-conexão, o UDP (*User Datagram Protocol*) [Postel, 1980]. Outra medida de projeto foi manter o SNMP tão independente quanto possível de outros serviços da rede, e este é o motivo pelo qual a sua mais recente versão (o SNMPv3) possui um modelo de segurança auto-contido.

Atualmente, o ambiente no qual as operações de gerência são realizadas mudou drasticamente desde a época na qual o SNMP foi concebido. Os dispositivos de rede atuais possuem um poder computacional bem maior que os de outrora, sendo capazes de realizar tarefas muito mais complexas. Isso evidencia cada vez mais o problema da microgerência, gerado a partir da troca de um número relativamente grande de mensagens para a realização de uma determinada tarefa que, utilizando outra abordagem, demandaria apenas uma ou duas mensagens.

Em [Sprenkels e Martin-Flatin, 1999], uma das críticas mais comuns ao SNMP é descrita. Em particular, ela diz respeito à capacidade de transferência volumosa de dados surgida a partir do SNMPv2, conforme comentado no Capítulo 2. Segundo aquela referência, esse requisito foi, na verdade, apenas parcialmente cumprido a partir do surgimento da PDU GetBulkRequest. Isso é devido ao fato de que o verdadeiro fator limitante para a transferência volumosa de dados é o tamanho do segmento UDP. Assim, se o tamanho de uma mensagem SNMP contendo uma PDU exceder o tamanho do campo de dados de um segmento UDP (que é de aproximadamente 64 KBytes), aquela mensagem não vai poder ser enviada indepentemente da natureza da PDU encapsulada.

Na esteira da crítica descrita no parágrafo anterior está uma outra: a de que não há necessidade do SNMP ater-se a um protocolo como o UDP. De fato, em [Wellens e Auerbach, 1996] é dito que a grande maioria das atividades de gerência de redes ocorrem quando a rede não apresenta problemas relacionados à sua conectividade física. Afirma-se também que, contra falhas daquela natureza, nenhum protocolo de transporte é eficiente, quer seja ele orientado a conexão ou não. Por fim, afirma-se que, em situações relacionadas a congestionamentos, algoritmos de controle empregando o TCP [Postel, 1981] são mais eficazes que segmentos UDP irregulares. Em [Sprenkels e Martin-Flatin, 1999] a adoção do TCP como protocolo de transporte já era sugerida, e, mais recentemente, em [Schönwälder, 2002] é definido um mapeamento do SNMP sobre o TCP, ainda que em caráter experimental.

Por fim, uma limitação fundamental do SNMP é a sua incapacidade de suportar transações, uma característica necessária à gerência de configuração. Conforme citado em [Pavlou et al., 2004], apesar de concebido para endereçar este tipo de problema, o COPS não mantém compatibilidade

com o SNMP e ainda herda parte de seus problemas, como a definição de um modelo de informações rudimentar, por exemplo.

7.3 Algumas abordagens revolucionárias

Uma vez que nesta Tese é proposta a utilização de JINI para a construção de NMSs, é interessante conhecer outras iniciativas que também contemplam o uso de tecnologias de âmbito geral para a construção daqueles sistemas. Essas iniciativas são classificadas em [Schönwälder et al., 2003] como *abordagens revolucionárias*. A utilização de JINI para a construção de NMSs pode, inclusive, ser enquadrada nessa categoria. As próximas subseções abordam algumas abordagens revolucionárias baseadas em tecnologias de grande repercussão.

7.3.1 Abordagens baseadas em CORBA

CORBA (*Common Object Request Broker Architecture*) é um padrão desenvolvido para interação entre Objetos em ambientes distribuídos [Vinoski, 1997]. Ele é especificada em [OMG, 2004a] pelo OMG (*Object Management Group*), um consórcio internacional fundado em 1989 e formado por mais de 600 membros que incluem fabricantes de sistemas de informações, desenvolvedores de software e usuários. O objetivo do OMG é a produção e manutenção de padrões industriais de interoperabilidade para aplicações voltadas à empresa.

A Figura 7.1 ilustra os principais componentes de CORBA. Essa tecnologia detalha as interfaces e característica do ORB (*Object Request Broker*), que é o componente definido no Modelo de Referência da OMA (*Object Management Architecture*). Esse Modelo é responsável por facilitar a comunicação entre Clientes e Implementações de Objetos. No Modelo de Objeto da OMA, o termo *Objeto* é definido como uma entidade com uma identidade distinta e imutável cujos Serviços podem ser acessados somente através de uma interface bem definida.

Referências para implementações de Objetos CORBA podem ser encontradas por Clientes mediante comunicação com os Serviços de Nomeação e de *Trading*, eles próprios Objetos CORBA e especificados em [OMG, 2004c] e [OMG, 2000], respectivamente. Utilizando o Serviço de Nomeação, Clientes obtêm referências para Objetos baseados nas interfaces que eles implementam, enquanto o fazem junto ao Serviço de *Trading* com base nas propriedades que esses Objetos apresentam.

A interface para um determinado Objeto é definida utilizando-se a OMG IDL (*Interface Definition Language*) [ISO/IEC e ITU-T, 1999], uma linguagem declarativa que atualmente possui mapeamento para várias linguagens de programação. Através de ferramentas apropriadas, essas interfaces dão origem a *stubs* (para Clientes) e *skeletons* (para as implementações de Objetos).

Um *stub* é um componente que efetivamente cria e emite solicitações no interesse do Cliente, enquanto um *skeleton* é um componente que entrega essas solicitações às implementações de Objetos. A utilização de *stubs* e *skeletons* é chamada de invocação estática, na qual esses componentes

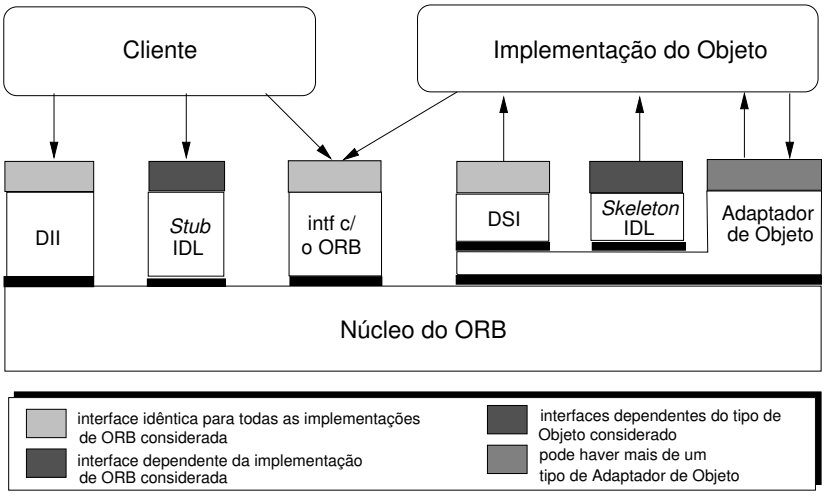

Figura 7.1: Principais componentes de CORBA.

são conectados diretamente aos Clientes e às implementações de Objetos, respectivamente. Eles têm, portanto, um conhecimento *a priori* das interfaces dos Objetos invocados.

Além desse tipo de invocação, CORBA disponibiliza também um mecanismo de invocação dinâmica, mediante o emprego da DII (*Dynamic Invocation Interface*) por Clientes e da DSI (*Dynamic Skeleton Interface*) por implementações de Objetos. Quando Clientes utilizam a DII, o ORB acessa transparentemente o Repositório de Interfaces (ele próprio um Objeto CORBA) para buscar informações sobre as interfaces implementadas pelos Objetos. A interface análoga à DII é a DSI, que permite às implementações de Objetos serem escritas sem ter *skeletons* compilados estaticamente com eles.

Outro componente de CORBA é o Adaptador de Objeto, responsável pelo registro, geração da referência e ativação de Objetos. O Adaptador de Objeto também é responsável pela entrega de solicitações repassadas pelo ORB às implementações de Objetos. O principal Adaptador de Objeto especificado pelo OMG é denominado POA (*Portable Object Adapter*).

O GIOP (*General Inter-ORB Protocol*) especifica a sintaxe de transferência e um conjunto padrão de formatos de mensagem para interoperação entre ORBs sobre qualquer protocolo de transporte orientado à conexão. O IIOP (*Internet Inter-ORB Protocol*) especifica como o GIOP é construído sobre TCP/IP. Com efeito, o relacionamento entre IIOP e GIOP é semelhante àquele verificado entre a implementação da interface de um Objeto e a sua definição.

Esforços dentro do OMG são classificados em esforços *verticalmente orientados*, cobrindo domínios específicos, e esforços *horizontalmente orientados*, independentes de qualquer domínio. Dentre os esforços verticalmente orientados, pode-se citar o emprego de CORBA em áreas como medicina, telecomunicações e negócios, por exemplo. Dentre os esforços horizontalmente orientados, pode-se citar as iniciativas relacionadas à passagem de Objetos por valor entre aplicações CORBA e a definição de um conjunto de serviços para monitoração e gerência de sistemas distribuídos.

Em [Pavlou, 1997], CORBA é citada como a contrapartida prática do ODP (*Open Distributed Processing*) [ISO/IEC e ITU-T, 1993], o modelo de referência proposto pela ISO para a construção de sistemas distribuídos abertos. Nessa referência, é introduzida uma arquitetura de gerência que retém as características operacionais da Gerência de Sistemas OSI [ISO/IEC e ITU-T, 1991] sobre um ambiente de processamento distribuído baseado em CORBA.

A proposta descrita em [Pavlou, 1997] cita a JIDM (*Joint Inter-Domain Management*), uma tecnologia que define de que forma componentes baseados na Gerência de Sistemas OSI ou no SNMP podem interoperar com componentes baseados em CORBA [The Open Group et al., 2000].

A JIDM possui atualmente duas especificações. A *Tradução de Especificação* descreve um mapeamento estático que estabelece equivalências entre a OMG IDL e as sintaxes GDMO (*Guidelines for Definition of Managed Objects*) [ISO/IEC e ITU-T, 1992] e SMI. Já a *Tradução de Interação* define um mapeamento dinâmico, estabelecendo como CORBA pode ser utilizada para a realização de operações análogas àquelas realizadas usando-se CMIP (*Common Management Information Protocol*) [ISO/IEC e ITU-T, 1998] ou SNMP.

Tomando como base uma versão da Tradução de Especificação emitida em 1994, [Pavlou, 1997] define *Brokers* de Gerência responsáveis por *clusters* de MOs, modelados como Objetos CORBA. Tais *Brokers* desempenham os papéis de fábricas de MOs e Serviços de Nomeação, *Trading* e Notificações no escopo daqueles *clusters*, provendo funcionalidades análogas àquelas providas pelo CMISE (*Common Management Information Service Element*), e suportando inclusive critérios de seleção de MOs baseados no fornecimento de parâmetros de escopo e filtragem.

A arquitetura apresentada em [Pavlou, 1997] é posteriormente refinada em [Pavlou, 1999], sendo justificada pelo fato de que a TMN (*Telecommunications Management Network*) [ITU-T, 1992] utiliza a Gerência de Sistemas OSI. Nessa linha, [Pavlou, 1999] apresenta propostas relacionadas ao surgimento de uma TMN totalmente baseada em CORBA, ou a estratégias de migração em direção àquela estratégia. Tais propostas são revisitadas, e validadas por protótipos, em [Ranc et al., 2000].

Outra iniciativa no estudo da utilização de CORBA na construção de NMSs é introduzida em [Pavon, 1999]. Aquele estudo baseia-se também na modelagem de MOs como Objetos CORBA, e na adoção facultada de *Brokers* de Gerência[1]. Em outras palavras, as aplicações poderiam gerenciar um determinado sistema, representado por vários MOs modelados como Objetos CORBA, de duas maneira distintas: ou interagindo diretamente com aqueles MOs, ou utilizando o *Broker* de Gerência.

Na proposta introduzida em [Pavon, 1999], as aplicações contactam inicialmente os Serviços de Nomeação e de *Trading* em busca de referências para um determinado MO diretamente, ou de um *Broker* que dá acesso àquele MO. Se uma referência direta para o MO é retornada, tais aplicações podem invocar operações do tipo `get`, `cancelGet`, `set` ou `action` junto àquele MO. Se uma referência para o *Broker* é retornada, operações do tipo `create` e `delete` podem também ser invocadas junto àquele componente.

Também é incentivado o uso dos vários Serviços disponibilizados por CORBA (tais como os

[1]Naquele estudo, porém, é considerada também a adoção de *gateways* CORBA/CMIP, conforme proposto na Especificação de Interação da JIDM emitida em 1997.

Serviços de Notificações, Segurança e Transações, dentre outros), como forma de prover tais funcionalidades aos NMSs construídos com um máximo de produtividade. A descrição da proposta introduzida em [Pavon, 1999] é finalizada com uma discussão sobre sua aplicação em TMN e na TINA (*Telecommunications Information Networking Architecture*) [Chapman e Montesi, 1995].

7.3.2 Abordagens baseadas em Web Services

Uma das tecnologias mais citadas atualmente para o desenvolvimento de aplicações distribuídas é *Web Services* [Booth et al., 2004]. A padronização de seus vários aspectos está a cargo de organizações como W3C (*World Wide Web Consortium*) e OASIS (*Organization for the Advancement of Structured Information Standards*), dentre outras. Conforme é ilustrado pela Figura 7.2, sua arquitetura básica é composta por uma série de blocos funcionais.

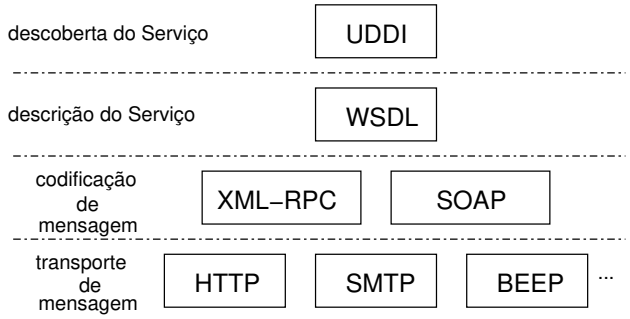

Figura 7.2: Principais componentes da arquitetura Web Services.

As interfaces implementadas pelos Serviços são descritas utilizando a WSDL (*Web Services Description Language*) [Chinnici et al., 2005]. Arquivos WSDL contêm as operações suportadas pelo Serviço, os parâmetros utilizados nestas operações e seus valores de retorno. Esses arquivos contêm ainda as mensagens utilizadas para a realização daquelas operações, a especificação da codificação a ser empregada nestas mensagens e a localização do Serviço. Essa última é expressa na forma de uma URI (*Uniform Resource Identifier*).

As mensagens podem ser codificadas utilizando-se protocolos como o XML-RPC e o SOAP. O XML-RPC [Winer, 1999] é um protocolo que utiliza a XML para realização de RPCs (*Remote Procedure Calls*). Nessa estratégia, as solicitações são obrigatoriamente enviadas utilizando-se o método POST do HTTP, as respostas sendo encapsuladas no corpo da mensagem de resposta desse protocolo. Outra alternativa de codificação é o SOAP (*Simple Object Access Protocol*) [Mitra, 2003], um protocolo mais complexo que o XML-RPC, mas que suporta uma variedade de protocolos para transporte de suas mensagens, tais como HTTP, SMTP (*Simple Message Transfer Protocol*) [Postel, 1982] ou BEEP (*Blocks Extensible Exchange Protocol*) [Rose, 2001], dentre outros.

A publicação e descoberta de Serviços são suportados através de um Serviço "meta" denominado UDDI (*Universal Description, Discovery, and Integration*) [Clement et al., 2004]. Esse Serviço

é definido pelo consórcio OASIS, não sendo formalmente reconhecido pelo W3C. Por isto, em alguns casos ele não é considerado como parte da arquitetura básica Web Services.

As informações acerca de um Web Service são publicadas em um registro UDDI em três níveis distintos. No nível das *White Pages*, são publicadas informações relativas a endereços de contatos e identificadores conhecidos da companhia provedora (fabricante) do Web Service. No nível das *Yellow Pages*, o Web Service é categorizado industrialmente com base em uma taxionomia padrão. Finalmente, no nível das *Green Pages* são postas informações técnicas acerca do Web Service, como a sua interface implementada, por exemplo.

Embora Web Services não seja uma tecnologia especificamente projetada para gerência de redes, algumas iniciativas têm surgido recentemente no intuito de empregá-la nesta área. Essas iniciativas foram formalmente encorajadas em eventos como o *workshop* organizado pelo IAB em junho de 2002 para a discussão de direções futuras para a área de gerência de redes [Schönwälder, 2003].

Naquele evento, seus participantes chegaram a um consenso de que o foco das pesquisas em gerência de redes deve ser posto em abordagens revolucionárias, em oposição às abordagens evolucionárias. Isso fez com que alguns Grupos de Trabalho que possuíam foco em abordagens evolucionárias, tais como o EoS (*Evolution of SNMP*) WG [IETF, 2002a] e o SMIng (*Next Generation Structure of Management Information*) WG [IETF, 2002b], fossem descontinuados. Por outro lado, propostas baseadas em XML foram incentivadas. Exemplos de propostas dessa natureza podem ser encontradas em [Choi et al., 2003] e [Strauss e Klie, 2003].

Haja vista Web Services ser a mais promissora tecnologia baseada em XML, estudos visando aplicá-la na área de gerência de redes surgiram naturalmente. Um desses estudos é descrito em [Schönwälder et al., 2003], que sugere três estratégias para o emprego de Web Services na construção de NMSs. Na primeira delas, uma aplicação desempenhando o papel de Gerente contacta um Web Service desempenhando o papel de Agente, invocando operações simples e utilizando o HTTP como protocolo de transporte. Em outra estratégia, este Web Service envia notificações àquele Gerente. A terceira e última estratégia contempla a invocação de operações complexas junto ao Web Service, objetivando a realização de cálculos estatísticos, atualização de tabelas de roteamento ou balanceamento de carga, por exemplo.

Em [Schönwälder et al., 2003], também é ressaltada a necessidade da utilização de transações em operações combinadas com vistas à produção de mudanças consistentes na configuração de dispositivos. Para tal, faz-se menção à especificação BPEL4WS (*Business Process Execution Language for Web Services*) [Andrews et al., 2003], produzida por empresas como Microsoft e IBM, dentre outras.

Duas abordagens extremas para a definição de interfaces implementadas por Web Services voltados à gerência de redes são também consideradas em [Schönwälder et al., 2003]. Na primeira delas os arquivos WSDL definem apenas um conjunto de operações básicas, tais como *get* e *set*. Nesta abordagem, os parâmetros das operações são passados como tipos opacos, o que significa que os arquivos WSDL não especificam ou interpretam os tipos dos vários MOs aos quais as operações destinam-se. É possível, no entanto, definir estes tipos utilizando *schemas* XML de mais alto nível.

Na outra abordagem, os arquivos WSDL definem operações separadas para cada MO, como `getIfInOctets` e `changeIfOperationalStatus` por exemplo. Nessa abordagem não há nenhuma

necessidade de definir-se *schemas* XML adicionais, porque os arquivos WSDL já contêm todas as informações necessárias para a gerência de um dispositivo. As operações citadas antes, por exemplo, possuiriam o parâmetro `ifIndex`, definindo a interface à qual relacionam-se.

Em [Sloten et al., 2004], é apresentado um estudo um pouco mais refinado sobre a definição de interfaces implementadas por Web Services voltados à gerência de redes. Naquele estudo, são apresentadas não apenas duas, mas quatro abordagens extremas para a definição daquelas interfaces. Essas abordagens caracterizam-se pela variação do nível de granularidade utilizado na definição das operações e pela maneira como seus parâmetros são definidos.

Mais especificamente, tanto para operações definidas utilizando um baixo nível de granularidade (por exemplo, `get`), quanto para aquelas definidas utilizando-se o maior nível de granularidade (por exemplo, `getInOctets`), seus parâmetros podem ser mantidos opacos (sendo definidos em *schemas* XML posteriores) ou serem transparentes (nesse caso sendo definidos no nível do arquivo WSDL da interface). Conforme pode-se observar, as abordagens descritas em [Schönwälder et al., 2003] são contempladas pela proposta apresentada em [Sloten et al., 2004].

Alguns protótipos de NMSs baseados em Web Services são descritos em [Drevers et al., 2004]. Esses protótipos foram projetados de forma a permitir a coleta de parâmetros modelando os MOs pertencentes à tabela `ifTable`. A coleta é feita de quatro maneiras distintas, cada uma delas ficando a cargo de um protótipo específico: por elemento da tabela, por linha, por coluna e, finalmente, a tabela como um todo. A intenção deste estudo foi avaliar o desempenho de cada um daqueles protótipos com relação ao tráfego gerado, bem como investigar o efeito da compressão sobre as mensagens transmitidas. Sua conclusão foi de que o emprego da compressão atenua o aumento de tráfego gerado a medida que mais parâmetros da tabela são coletados.

Em [Pavlou et al., 2004], o emprego de Web Services para gerência de redes é comparado com SNMP e CORBA. Protótipos foram construídos para medição do tempo de resposta e do tráfego gerado empregando cada uma dessas tecnologias. O resultado mostrou que Web Services, por apresentar mensagens codificadas em XML, gerou um tráfego muito maior que aqueles gerados pelas outras abordagens. Em particular, tomando como base os protótipos desenvolvidos, o tráfego gerado utilizando-se Web Services foi cerca de oito vezes superior àquele verificado utilizando-se CORBA. Os tempos de resposta para os protótipos baseados em Web Services também mostraram-se geralmente superiores àqueles obtidos para os demais protótipos, essa diferença sendo atribuída ao tempo de codificação relativamente maior das mensagens utilizando XML.

Outra comparação entre Web Services e SNMP é realizada em [Pras et al., 2004]. Neste estudo, porém, um número bem maior de Agentes SNMP foram utilizados, e vários protótipos utilizando Web Services foram construídos. Os parâmetros investigados foram o tráfego gerado, consumo de CPU e memória e tempo de resposta. Os resultados obtidos mostraram mais uma vez que o tráfego gerado pelos protótipos baseados em Web Services é superior àquele obtido com a utilização de Gerentes e Agentes SNMP. Um fato interessante, porém, é que a situação inverte-se quando compressão é aplicada. Neste caso, quando um número grande de MOs é coletado, o tráfego Web Services comprimido é inferior ao tráfego SNMP comprimido. O efeito da compressão como forma de diminuir o tráfego gerado por Web Services já tinha sido proposto em [Pavlou et al., 2004], embora não tenha sido investigado naquele estudo.

Os testes realizados em [Pras et al., 2004] mostraram mais uma vez que o tempo necessário para a codificação de mensagens SNMP, utilizando-se as BER (*Basic Encoding Rules*), é superior ao tempo necessário para codificação das mensagens Web Services utilizando-se XML. Além disso, mostrou-se que, quando a compressão é aplicada, a diferença entre estes tempos de codificação aumenta mais ainda. No caso do consumo de memória, verificou-se que os protótipos baseados em Web Services construídos apresentaram um consumo inferior àquele verificado para os componentes baseados em SNMP. Por fim, observou-se que os tempos de resposta obtidos em ambas as abordagens foram muito semelhantes. Conquanto o impacto gerado por mecanismos que provejam segurança e suporte à transações não tenham sido investigados nesse estudo, a conclusão obtida em [Pras et al., 2004], bem menos reticente que a observada em [Pavlou et al., 2004], é a de que não há razão convincente para refutar a utilização de Web Services para a construção de NMSs.

7.4 Comparações entre JINI e as demais tecnologias

Diversos critérios podem ser utilizados em comparações de âmbito geral envolvendo JINI e tecnologias como CORBA e Web Services. Porém, quando considera-se a utilização dessas tecnologias para a construção de NMSs, alguns destes critérios merecem destaque.

Para realizar tal comparação, essa seção inicialmente descreve como estes critérios são atendidos por cada uma delas. Em particular, os critérios escolhidos relacionam-se aos aspectos de modelagem de Serviços, utilização de infra-estrutura existente, possibilidade de emprego de Interfaces com o Usuário flexíveis e a aplicabilidade das tecnologias em dispositivos com baixo poder computacional. Todos estes critérios são cobertos por subseções específicas.

Em seguida, uma análise é realizada com base nas informações obtidas naquelas subseções. Seu objetivo é definir o quão adequada é JINI para a construção de NMSs, quando comparada às demais tecnologias.

7.4.1 Modelagem de Serviços

No Capítulo 4, a primeira das considerações gerais introduzidas com relação à utilização de JINI na construção de NMSs foi a associação de Serviços JINI a sistemas gerenciados. Conforme foi visto, vários níveis de granularidade podem ser empregados naquela modelagem, sendo o maior deles referente à modelagem de um Serviço correspondente a um MO definido segundo as GDMO.

Foi visto também que a adoção deste nível de granularidade na modelagem poderia resultar num problema sério de escalabilidade. Esse mesmo problema é observado em [Pavlou et al., 2004], quando considera-se o emprego de Objetos CORBA na gerência de redes. Naquela referência, recomenda-se a adoção de uma técnica de modelagem diferente da JIDM para emprego de Objetos CORBA em aplicações voltadas à gerência de redes.

Por exemplo, atributos comumente acessados devem ser retornados coletivamente através de um método apropriado. O mesmo é válido para entidades dinâmicas tais como conexões, que não

devem ser modeladas através de Objetos/interfaces separados. Essas recomendações foram também direcionadas para o emprego de Web Services naquele cenário específico.

Assim, de forma a melhor ilustrar como Serviços funcionalmente equivalentes podem ser projetados com base em cada uma das tecnologias citadas até então, alguns protótipos foram construídos seguindo-se a metodologia descrita em [Pavlou et al., 2004], e tempos de invocação e tráfegos gerados a partir destes protótipos foram medidos.

Para efeito de comparação, considere-se inicialmente um conjunto de MOs definidos segundo a SMI do SNMP. Os MOs escalares presentes neste conjunto são aqueles cinco pertencentes ao grupo `system` da MIB-II e citados no Capítulo 5. Ou seja, são os MOs `sysDescr`, `sysName`, `sysLocation`, `sysContact` e `sysUpTime`. Dentre os MOs tabulares, estão aqueles pertencentes à tabela `prTable`, também citada no Capítulo 5. Em particular, estes MOs encontram-se descritos na Tabela 7.1.

MO	Descrição
`prIndex`	índice de referência para cada processo observado.
`prNames`	nome do processo de interesse.
`prMin`	a quantidade mínima de instâncias do processo que devem estar em execução.
`prMax`	a quantidade máxima de instâncias do processo que devem estar em execução.
`prCount`	quantidade de instâncias do processo em execução.
`prErrorFlag`	uma *flag* de erro. Inicialmente configurada com o valor 0, assume o valor de 1 quando o número de instâncias do processo em execução for menor que o mínimo estabelecido ou maior que o máximo estabelecido.
`prErrMessage`	uma mensagem de erro descrevendo o problema (quando ele existe).
`prErrFix`	quando configurada em 1, esta *flag* indica que o erro descrito pelo MO acima deve disparar a execução do comando (ou *script*) descrito pelo próximo MO.
`prErrFixCmd`	descreve o comando (ou *script*) que deve ser invocado em virtude do erro.

Tabela 7.1: MOs presentes na tabela *prTable*.

Como Agente SNMPv3 suportando esses MOs, utilizou-se o aplicativo `snmpd` distribuído com a versão 5.2.2 do *toolkit* NET-SNMP. Protótipos de Gerentes foram construídos com base nas versões 1.4.2_06 do Java SDK e 3.3 da API SNMP provida pela AdventNet. Esses protótipos possibilitam a coleta de um ou todos MOs escalares, utilizando uma única PDU GetRequest, e a coleta da tabela `prTable`. Essa última pode ser realizada utilizando consecutivas PDUs GetNextRequest, que retornam uma linha por vez, ou através de uma única PDU GetBulkRequest retornando todas as linhas da tabela.

Com relação às tecnologias investigadas, tem-se, para CORBA, um Objeto desenvolvido com base na interface ilustrada a seguir. Esse Objeto, bem como seus respectivos Clientes, foram construídos com o auxílio da API CORBA suportada pela versão 1.4.2_06 do Java SDK.

```
module WorkStation {

  struct System {
    string sysDescr;
```

```
    long long sysUpTime;
    string sysContact;
    string sysName;
    string sysLocation;
  };

  struct PrRow {
    long prIndex;
    string prNames;
    long prMin;
    long prMax;
    long prCount;
    long prErrorFlag;
    string prErrMessage;
    long prErrFix;
    string prErrFixCmd;
  };

  struct PrTable {
    sequence<PrRow, 40> rows;
  };

  interface WorkStationService {

    string getSysDescr();
    long long getSysUpTime();
    string getSysContact();
    string getSysName();
    string getSysLocation();

    System getSystem();

    PrRow getRow(in long index);

    PrTable getTable();
  };
};
```

Conforme pode-se observar, essa interface, que é definida segundo a OMG IDL, permite coletar parâmetros análogos a cada um dos cinco MOs escalares citados há pouco. Isso pode ser feito individualmente (utilizando os métodos `getSysDescr`, `getSysUpTime`, `getSysContact`, `getSysName` e `getSysLocation`) ou coletivamente (utilizando o método `getSystem`). Ela permite também a obtenção de uma tabela análoga à `prTable`, através de invocações consecutivas do método `getPrRow` ou utilizando apenas uma invocação do método `getPrTable`. O método `gerPrRow` permite obter uma, dentre as potencialmente muitas, linhas que compõem a tabela, e recebe como parâmetro de entrada a identificação desta linha.

Usando as versões 1.4.2_06 do Java SDK e 1.3 do Axis [The Apache Software Foundation, 2005], esse último um utilitário para a construção de aplicações utilizando SOAP, foram construídos um

Web Service e os seus respectivos Clientes. Um trecho do arquivo WSDL que descreve o Serviço é ilustrado abaixo (a listagem completa deste arquivo encontra-se no Apêndice C)[2].

```
<?xml version="1.0" encoding="UTF-8"?>
 ...
 <wsdl:types>
  ...
   <complexType name="System">
    <sequence>
     <element name="sysDescr" nillable="true" type="xsd:string"/>
      ...
     <element name="sysUpTime" type="xsd:long"/>
    </sequence>
   </complexType>
   <complexType name="PrRow">
    <sequence>
     <element name="prIndex" type="xsd:int"/>
      ...
     <element name="prErrFixCmd" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="PrTable">
    <sequence>
     <element name="rows" nillable="true" type="impl:ArrayOf_tns1_PrRow"/>
    </sequence>
   </complexType>
  ...
 </wsdl:types>
 <wsdl:message name="getSysDescrRequest">
 </wsdl:message>
 <wsdl:message name="getSysDescrResponse">
    <wsdl:part name="getSysDescrReturn" type="soapenc:string"/>
 </wsdl:message>
  ...
 <wsdl:portType name="WorkStationServer">
    <wsdl:operation name="getSysDescr">
       <wsdl:input message="impl:getSysDescrRequest"
          name="getSysDescrRequest"/>
       <wsdl:output message="impl:getSysDescrResponse"
          name="getSysDescrResponse"/>
    </wsdl:operation>
     ...
 </wsdl:portType>
 <wsdl:binding name="WorkStationServerSoapBinding"
    type="impl:WorkStationServer">
  ...
```

[2]As definições neste arquivo seguem a versão 1.1 da WSDL. Na versão atual (2.0), os elementos $< porttype >$ e $< port >$ são substituídos pelos elementos $< interface >$ e $< endpoint >$, respectivamente.

```
 </wsdl:binding>
 <wsdl:service name="WorkStationServerService">
    <wsdl:port binding="impl:WorkStationServerSoapBinding"
       name="WorkStationServer">
       <wsdlsoap:address location="http://143.106.50.214:8080/axis/
          services/WorkStationServer"/>
    </wsdl:port>
 </wsdl:service>
```

Conforme pode-se observar, o arquivo WSDL define quais são as operações disponibilizadas pelo Serviço, e quais são os tipos e as mensagens utilizados para a realização destas operações. O arquivo define ainda o mecanismo utilizado para a codificação destas mensagens (SOAP) e a localização do Serviço.

Tomando-se como base a proposta apresentada em [Sloten et al., 2004], a interface descrita possui operações definidas em um nível de granularidade elevado, permitindo a obtenção individual de parâmetros análogos a cada um dos cinco MOs escalares citados há pouco. No entanto, há também operações definidas em níveis de granularidade menores, permitindo a obtenção coletiva daqueles parâmetros e a obtenção de uma tabela análoga a `prTable`, essa última podendo ser realizada seqüencialmente ou de uma vez só. Outra característica na definição da interface é que seus parâmetros relativos às operações são definidos de forma transparente.

Por fim, procedeu-se à construção de Serviços JINI funcionalmente similares àqueles descritos até então. Foram construídos dois Serviços: um deles foi desenvolvido com base na versão 1.2_002 do JTSK, enquanto o outro foi construído utilizando a versão 2.0_002 desse *toolkit*. Esses Serviços comunicam-se com seus Clientes mediante uso dos protocolos JRMP e JERI, respectivamente. Ambos implementam a interface ilustrada abaixo.

```
package WorkStation;

import java.rmi.*;

public interface WorkStationService extends Remote {

    String getSysDescr() throws RemoteException;
    long getSysUpTime() throws RemoteException;
    String getSysContact() throws RemoteException;
    String getSysName() throws RemoteException;
    String getSysLocation() throws RemoteException;

    WorkStation.System getSystem() throws RemoteException;
    WorkStation.PrRow getRow(int index) throws RemoteException;
    WorkStation.PrTable getTable() throws RemoteException;
}
```

Todos os Serviços descritos até então foram instalados em um PC equipado com um processador AMD K6-II executando a 550 MHz e possuindo 128 MBytes de RAM. Nesta máquina, encontra-se instalada a versão 7.0 da distribuição Linux Conectiva. Por outro lado, os Clientes desses Serviços

foram instalados em um outro PC equipado com um processador Pentium III executando a 550 MHz e possuindo 128 MBytes de RAM. Nessa outra máquina, encontra-se instalada a versão 5.0 da distribuição Linux Conectiva. Ambas as máquinas encontram-se conectadas através de um enlace ponto-a-ponto *Ethernet*.

Com base nos Serviços citados até então e seus respectivos Clientes, procedeu-se às medidas dos tempos de invocação para cada uma das tecnologias. Essas medidas contemplaram cenários onde foram exploradas todas as funcionalidades oferecidas pelos Serviços desenvolvidos[3]. O gráfico construído com base nos valores obtidos é ilustrado pela Figura 7.3.

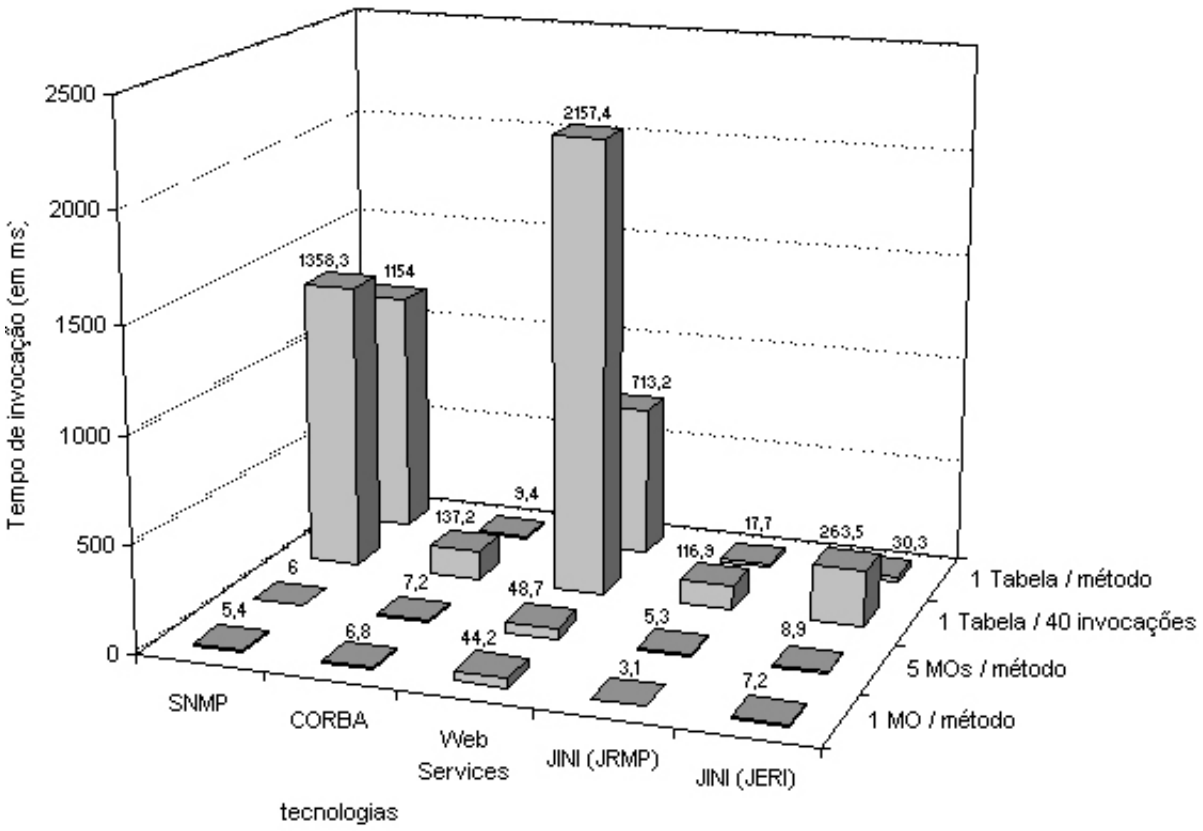

Figura 7.3: Tempos de invocação obtidos em cada um dos protótipos construídos.

Conforme pode ser visto na Figura anterior, no que concerne à coleta do MO escalar `sysDescr`, ou de um parâmetro que o modele, o menor tempo de invocação foi obtido com o protótipo baseado em SNMP. Utilizando este protótipo, o efeito de adicionar os MOs `sysName`, `sysLocation`,

[3]Para cada um dos testes descritos ao longo desta Seção, considera-se um conjunto de dez amostras, cada uma contendo o resultado de 100 repetições. Cada valor ilustrado corresponde à média obtida na amostra que apresenta menor desvio padrão, ou seja, é a média obtida na amostra mais homogênea.

`sysContact` e `sysUpTime` ao MO escalar `sysDescr` numa mesma PDU GetRequest resultou num aumento de 11,11 % no tempo de invocação.

Levando-se em conta os mesmos quesitos, mas considerando agora somente as abordagens revolucionárias, o menor tempo de invocação para a coleta de um parâmetro modelando o MO escalar `sysDescr` foi obtido com o protótipo baseado em JINI utilizando JRMP (3,10 ms), seguido do protótipo baseado em CORBA (6,80 ms) e do protótipo baseado em JINI utilizando JERI (7,20 ms). Embora apresentem um desempenho satisfatório neste quesito, os protótipos baseados em JINI foram os que experimentaram os maiores acréscimos percentuais quando parâmetros modelando os demais MOs escalares foram também coletados coletivamente utilizando um método apropriado. Esses acréscimos corresponderam a 70,97 % e 23,61 % dos valores obtidos para a coleta de apenas um parâmetro utilizando JRMP e JERI, respectivamente.

Os menores acréscimos percentuais foram obtidos com os protótipos baseados em CORBA (5,88 %) e Web Services (10,18 %). Em [Pavlou et al., 2004], é dito que esse acréscimo percentual é uma função linear do tempo necessário para codificação/decodificação. Assim, para os protótipos baseados em JINI utilizando JRMP esse tempo é aproximadamente seis vezes superior àquele obtido com o protótipo baseado em SNMP, enquanto que o protótipo utilizando JERI essa proporção cai para aproximadamente duas vezes. Por outro lado, os tempos de codificação/decodificação obtidos com os protótipos baseados em CORBA e Web Services mostraram-se 47 % e 8 % inferiores, respectivamente, àquele obtido com o protótipo baseado em SNMP. Indepentemente desses últimos comentários, os tempos de invocação obtidos com os protótipos baseados em JINI mostram que essas diferenças não são preocupantes, pelo menos para os protótipos considerados.

No que diz respeito à coleta de uma tabela inteira de MOs composta por 40 linhas (ou à coleta de uma representação dessa tabela), os resultados ilustrados pelo gráfico da Figura 7.3 mostram que a obtenção linha-a-linha dessa tabela (ou de sua representação) resultou num tempo de invocação superior àquele obtido quando ela é coletada de uma vez só.

Além disso, todos os protótipos baseados em abordagens revolucionárias apresentaram reduções percentuais no tempo de invocação maiores que os 15,04 % obtidos com o protótipo baseado em SNMP quando, naquele protótipo, uma única PDU GetBulkRequest é utilizada em detrimento de várias PDUs GetNextRequest. De fato, a maior redução percentual observada quando a tabela é coletada com o auxílio de um único método foi obtida com o protótipo baseado em CORBA (93,15 %), seguido do protótipo baseados em JINI utilizando JERI (88,50 %). O protótipo baseado em JINI utilizando JRMP obteve uma redução percentual de 84,86 %, e no protótipo baseado em Web Services esta redução foi de 66,94 %. Os tempos de invocação obtidos por cada protótipo encontram-se sumarizados na Tabela 7.2.

A Figura 7.4 ilustra o tráfego gerado pelos protótipos construídos, em cada um dos testes há pouco citados. Essas medições tomaram como base as PDUs do nível de enlace de dados (quadros) exclusivamente transmitidas por ocasião da invocação de métodos ou lançamento de PDUs SNMP. Elas foram realizadas com a ajuda da versão 0.8.11 do aplicativo `ethereal` [Ethereal, Inc., 2005].

Conforme pode-se observar, a troca de PDUs GetRequest/Response visando a coleta do MO escalar `sysDescr` gerou um tráfego de 370 Bytes. No que concerne os protótipos baseados em abordagens revolucionárias, todos geraram tráfego superior a esse valor na invocação de um método

Tecnologia	1 MO/método	5 MOs/método	1 Tabela/40 métodos	1 Tabela/método
SNMP	5,40	6,00	1.358,30	1.154,00
CORBA	6,80	7,20	137,20	9,40
Web Services	44,20	48,70	2.157,40	713,20
JINI (JRMP)	3,10	5,30	116,90	17,70
JINI (JERI)	7,20	8,90	263,50	30,30

Tabela 7.2: Tempos de invocação obtido (em ms) para cada uma dos protótipos construídos.

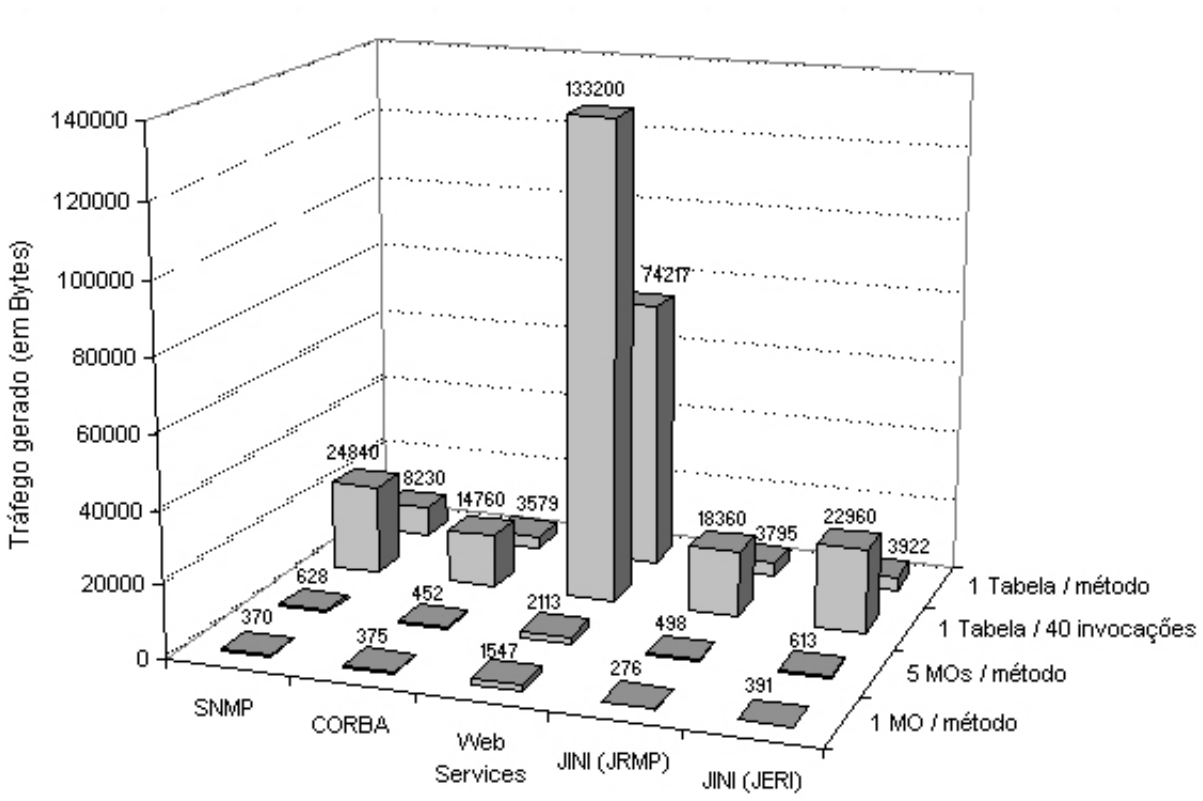

Figura 7.4: Tráfegos gerados por cada um dos protótipos.

cujo retorno era um parâmetro modelando aquele MO. A exceção foi o protótipo baseado em JINI utilizando JRMP, que gerou um tráfego de 276 Bytes. Por outro lado, quando novos MOs escalares (ou parâmetros que os modelam) foram adicionados, todos os protótipos geraram tráfegos inferiores

aos 628 Bytes gerados pela troca de PDUs SNMP. A exceção, nesse caso, foi o protótipo baseado em Web Services, que gerou um tráfego de cerca de 2 KBytes. Em particular, nesse quesito, o menor tráfego foi gerado pelo protótipo baseado em CORBA (452 Bytes).

Quando a coleta da tabela `prTable` é considerada, o tráfego gerado na sua coleta linha-a-linha é cerca de três superior ao tráfego gerado quando suas 40 linhas são retornadas utilizando-se uma única PDU GetBulkRequest. Neste quesito, quase todos os protótipos baseados em abordagens revolucionárias geraram tráfegos inferiores àqueles obtidos com o protótipo baseado em SNMP. A exceção neste caso foi o protótipo baseado em Web Services, que gerou tráfegos cerca de cinco e nove vezes superiores àqueles gerados pelo protótipo baseado em SNMP quando os parâmetros modelando a tabela `prTable` são coletados linha-a-linha e de uma vez só, respectivamente.

Na coleta de parâmetros modelando a tabela `prTable`, o menor tráfego gerado pela sua coleta linha-a-linha foi obtido com o protótipo baseado em CORBA (14,41 KBytes), seguido dos protótipos baseado em JINI utilizando JRMP (17,93 KBytes) e JERI (22,42 KBytes), e, por fim, do protótipo baseado em Web Services (130,08 KBytes). Quando esses parâmetros são coletados de uma vez só, os tráfegos gerados seguem a mesma seqüência: em primeiro lugar, está o protótipo baseado em CORBA (3,50 KBytes), seguidos dos protótipos baseados em JINI utilizando JRMP (3,50 KBytes) e JERI (3,70 KBytes) e, por fim, do protótipo baseado em Web Services (72,48 KBytes). Os tráfegos gerados por cada protótipo encontram-se sumarizados na Tabela 7.3.

Tecnologia	1 MO/método	5 MOs/método	1 Tabela/40 métodos	1 Tabela/método
SNMP	370	628	24.840	8.230
CORBA	375	452	14.760	3.579
Web Services	1.547	2.113	133.200	74.217
JINI (JRMP)	276	498	18.360	3.795
JINI (JERI)	391	613	22.960	3.922

Tabela 7.3: Tráfegos gerados (em Bytes) por cada um dos protótipos.

7.4.2 Utilização de infra-estrutura existente

Um fator decisivo para a difusão de uma tecnologia projetada para a construção de sistemas distribuídos é o montante de Serviços auxiliares agregados a ela. O conjunto desses Serviços é denominado aqui de *infra-estrutura*. Essa infra-estrutura pode ser disponibilizada como parte do modelo de programação da tecnologia, ou até mesmo ser desenvolvida separadamente, sendo posteriormente incorporada à mesma.

A ausência da definição de uma infra-estrutura associada a uma tecnologia faz com que desenvolvedores que a utilizam adotem soluções *ad-hoc* em seus sistemas, prejudicando a interoperabi-

lidade com outros sistemas. Isso contribui, a médio e longo prazo, para o abandono da tecnologia em detrimento de outras com maior infra-estrutura.

Na área de gerência de redes TCP/IP, especificamente, é necessário que NMSs construídos com base em abordagens revolucionárias utilizem a infra-estrutura associada às tecnologias nas quais baseiam-se essas abordagens. Essa necessidade vem do fato de que, haja vista tais abordagens terem a pretensão de substituir o modelo tradicional baseado no binômio SNMP/SMI, nada mais justo do que proverem funcionalidade análogas às providas pelo modelo tradicional.

Dentre os vários aspectos que compõem uma infra-estrutura, talvez os mais relevantes para a construção de NMSs sejam aqueles referentes ao registro e descoberta de Serviços e à utilização de eventos/notificações, segurança e transações.

Registro e descoberta de Serviços

Mecanismos referentes à descoberta dinâmica de Agentes SNMP não são cobertos por aquela tecnologia, esse procedimento devendo ser realizado numa base *ad-hoc* (provavelmente através do envio *multicast* de PDUs endereçadas ao *port* 162 de dispositivos). O mesmo pode ser dito com relação aos MOs suportados por aqueles Agentes, cujo conhecimento, em tempo de execução, pode ser viabilizado apenas por ferramentas do tipo navegadores de MIB.

Por outro lado, CORBA, Web Services e JINI possuem suporte para o registro e descoberta de seus respectivos Serviços. Conforme comentado na Subseção 7.3.1, CORBA conta com Serviços de Nomeação e de *Trading* para esses procedimentos. Web Services, de acordo com o comentado na Subseção 7.3.2, utiliza um mecanismo sofisticado para isso, o UDDI. Como foi visto no Capítulo 3, JINI utiliza o LUS, encarregado do registro e descoberta de Serviços.

Todas as tecnologias citadas no parágrafo anterior permitem que a descoberta de seus respectivos Serviços seja realizada utilizando-se também informações que vão além da interface implementada por eles. Essas informações podem estar contidas nas *propriedades* de Objetos CORBA, nas *White/Yellow pages* de Web Services ou nos *atributos* de um Serviço JINI.

Eventos/notificações

No tocante a eventos/notificações, a abordagem tradicional vale-se de PDUs *Trap* e *Inform* para a comunicação assíncrona entre Gerentes e Agentes SNMP. Essa estratégia segue o modelo *push*, no qual o supridor de um evento o transmite para os seus consumidores.

O Serviço de Evento definido para CORBA em [OMG, 2004b] provê uma API usada para transmissão de eventos entre Objetos distribuídos. Além de suportar o modelo *push*, esse Serviço permite também a utilização do modelo *pull*. Nesse modelo, o consumidor de um evento aguarda uma notificação informando que o mesmo está disponível, para em seguida coletá-lo junto ao seu respectivo supridor.

A especificação do Serviço de Evento define ainda o conceito de *Canais de Evento*, que nada mais são do que *brokers* que permitem a comunicação assíncrona entre múltiplos supridores e múltiplos consumidores de evento.

Especificado em [OMG, 2004d], o Serviço de Notificação CORBA estende o Serviço de Evento, porém preservando toda a semântica definida por este último. Dentre as funcionalidades adicionadas pelo Serviço de Notificação estão uma melhor estruturação dos eventos a serem transmitidos e a utilização de filtros.

A OASIS define uma família inteira de especificações relacionadas ao emprego de notificações em Web Services, denominada WSN (*Web Services Notification*). Em [Graham et al., 2005], são definidas as interfaces utilizadas por produtores e consumidores de Notificações. Mecanismos utilizados para classificação de eventos são definidos em [Vambenepe et al., 2005], e interfaces para *brokers* de Notificações são definidas em [Chappell e Liu, 2005].

Conforme citado na Seção 3.3, JINI possui suporte nativo ao uso de eventos. Seu modelo de programação define, desde as suas versões iniciais, uma estratégia do tipo *push* na qual Ouvidores de Evento registrados junto às aplicações geradoras transmitem esses eventos aos seus consumidores finais.

Segurança

Mecanismos auto-contidos, em particular o USM e o VACM, garantem ao modelo tradicional baseado no binômio SNMP/SMI a possibilidade do uso de autenticação e confidencialidade na troca de mensagens SNMP, bem como um controle de acesso aos MOs presentes na MIB suportada pelos Agentes.

O modelo de segurança definido para CORBA, também denominado de *CORBAsec*, é especificado em [OMG, 2002b]. Ele é capaz de prover identificação e controle de acesso na invocação remota de métodos, comunicação segura, não-repudiação, auditoria e administração. Esse modelo prevê sistemas de segurança que abrangem desde os Objetos definidos por usuários até a combinação de hardware e sistema operacionais locais, passando pelo núcleo do ORB e seus serviços.

A especificação CORBAsec em si não provê qualquer mecanismo de segurança. Ao invés disso, ela provê apenas uma arquitetura e interfaces padronizadas que podem ser utilizadas para dar segurança em um ambiente de Objetos distribuídos CORBA. Os mecanismos de segurança devem ser providos separadamente. Opções disponíveis para utilização são Kerberos, SPKM (*Simple Public Key GSS-API Mechanism*) [Adams, 1996] e SESAME (*Secure European System for Applications in a Multi-vendor Environment*) [Parker e Pinkas, 1995], por exemplo.

Em CORBAsec, dois níveis de segurança são providos. O primeiro nível, também chamado de nível do ORB, fornece segurança para aplicações que não têm ciência da segurança, garantindo controle de acesso aplicado pelo ORB e auditoria de eventos do sistema. O segundo nível, também conhecido como nível de aplicação, estende o primeiro nível com outras interfaces de aplicação e administração. O segundo nível é utilizado por aplicações cientes da segurança, permitindo que elas implementem-na de forma personalizada.

Várias especificações relacionadas ao emprego de segurança em Web Services foram (e continuam sendo) produzidas. A WSS (*Web Services Security*) [Nadalin et al., 2004], proposta pela OASIS, estabelece de que forma os padrões XMLDS (*XML Digital Signature*) [Bartel et al., 2002]

e Encriptação XML [Imamura et al., 2002], definidos em conjunto pelo W3C e pela IETF, podem ser aplicados em mensagens SOAP. Nesse contexto, o compartilhamento de informações de autenticação entre diferentes aplicações pode ser realizada através da SAML (*Security Assertion Markup Language*) [Cantor et al., 2005], também especificada pela OASIS.

A XKMS (*XML Key Management Specification*) [Hallam-Baker e MySore, 2005], produzida pelo W3C, define protocolos para registro e distribuição de chaves públicas, sendo sugerida sua utilização em conjunto com os padrões XMLDS e Encriptação XML. Já a XACML (*eXtensible Access Control Markup Language*), especificada pela OASIS em [Moses, 2005], define uma linguagem para estabelecimento de políticas de controle de acesso a recursos.

Conforme citado na Seção 3.4, JINI possui atualmente um modelo de segurança e componentes infra-estruturais capazes de utilizar mecanismos tais como SSL ou Kerberos para prover autenticação e confidencialidade nos dados trocados entre aplicações, bem como utilizar URLs HTTPMD para garantir a integridade do código descarregado dinamicamente a partir da rede.

Um detalhe interessante é que, no modelo de segurança definido para o SNMP (USM), a aplicação dos mecanismos de autenticação e confidencialidade (quando necessários), é feita no nível de mensagem. O mesmo comentário é válido para Web Services. A conseqüência disso é possibilidade de aplicação de procedimentos de autenticação e confidencialidade apenas em partes da mensagem, em oposição ao que é realizado quando tais mecanismos são aplicados no nível de invocação de métodos, como é o caso de JINI e CORBA.

Transações

Conforme comentado anteriormente nesse capítulo, uma das maiores deficiências observadas atualmente no modelo tradicional é a ausência de suporte a transações, de forma a garantir alterações consistentes na configuração dos equipamentos e serviços de rede. Felizmente, não apenas JINI, como também as demais tecnologias abordadas nesta Tese, provêem tal suporte ou possuem estudos para implementá-lo.

Suporte a transações ACID distribuídas em CORBA é provido pelo OTS (*Object Transaction Service*), especificado em [OMG, 2003]. Esse Serviço permite a utilização de transações simples e aninhadas, definindo conceitos tais como *Clientes e Objetos Transacionais* e *Objetos Recuperáveis e de Recurso*.

Clientes Transacionais originam transações junto ao Serviço de Transação e invocam operações em Objetos Transacionais. O comportamento dos Objetos Transacionais é afetado por invocações dentro do escopo de uma transação. Esses Objetos garantem a consistência de dados mediante comunicação com Objetos Recuperáveis, que registram Objetos de Recurso com o Serviço de Transações. Os Objetos de Recurso definem os dados cuja alteração está sob efeito da transação.

Um dos Comitês Técnicos da OASIS tem por objetivo a definição de mecanismos para o emprego de transações em Web Services. Esse Comitê tem trabalhado com base nas especificações [Cabrera et al., 2005a], [Cabrera et al., 2005b] e [Cabrera et al., 2005c]. Em [Cabrera et al., 2005a] é descrito um modelo geral para a coordenação de ações envolvendo vários Web Services com vistas a um objetivo comum. Protocolos de coordenação específicos a ambientes nos quais o pro-

cessamento é baseado em transações atômicas são definidos em [Cabrera et al., 2005b], enquanto em [Cabrera et al., 2005c] são definidos protocolos a serem utilizados em ambientes envolvendo atividades de negócios. Essas especificações estabelecem a associação com outros modelos, como o WSS por exemplo.

Quanto à JINI, foi visto na Seção 3.3 que essa tecnologia possui suporte ao uso de transações, desde as suas versões iniciais.

7.4.3 Emprego de Interfaces com o Usuário flexíveis

O Capítulo 4 ilustrou como usuários de uma Aplicação de Gerência podem efetuar a monitoração e o controle de um sistema gerenciado através de uma Interface especificamente projetada para essa finalidade.

Conforme descrito naquele capítulo, a realização de tais procedimentos somente é possível através da utilização, na construção de NMSs, de uma das iniciativas surgidas no âmbito da Comunidade JINI. Essa iniciativa, o Projeto *ServiceUI*, permite que um ou mais Objetos de Interface com o Usuário (ou Adaptadores de Protocolo) possam ser descarregados dinamicamente a partir do *proxy* para um Serviço, tornando possível a sua utilização através de formas que vão além do tradicional uso programático.

Funcionalidades análogas àquela não são encontradas de forma natural em *stubs* para Objetos CORBA ou Web Services. De fato, naquelas tecnologias a própria definição do termo *stub* é mais restritiva que a observada na definição de *proxies* para Serviços JINI. Em outras palavras, *stubs* para Objetos CORBA ou Web Services são tidos como componentes de software que permitem a invocação de métodos ou a troca de mensagens de forma programática, e nada mais.

Em CORBA, porém, Objetos de Interface com o Usuário poderiam ser empregados utilizando o conceito de *Interceptadores Portáveis* [OMG, 2001], que nada mais são do que Objetos genéricos que podem ser empregados no caminho normal percorrido em uma invocação remota. À exceção de JINI, porém, nenhuma outra abordagem utilizada para a gerência de redes (seja ela tradicional, evolucionária ou revolucionária) permite a utilização de um recurso tão poderoso de maneira tão natural. Essa limitação é decorrente do fato de que tais abordagens têm seu foco no protocolo utilizado na comunicação entre seus componentes.

Na abordagem proposta nesta Tese, o compromisso mantido entre os componentes de um NMS está em um nível de abstração mais elevado: a interface implementada pelo Serviço. Essa medida abre espaço para o surgimento de um conjunto de conjecturas acerca de como esse Serviço (de gerência) é melhor realizado. Conforme foi descrito, uma dessas conjecturas é a utilização do Projeto *ServiceUI*.

7.4.4 Aplicabilidade em dispositivos com baixo poder computacional

Um dos requisitos básicos por trás do surgimento do modelo tradicional baseado no binômio SNMP/SMI foi a capacidade de ser utilizado em dispositivos com baixo poder computacional. Mui-

to embora o hardware destes dispositivos tenha experimentado uma considerável evolução desde a introdução daquele modelo, as abordagens revolucionárias ainda devem ater-se àquele requisito.

Isso ocorre porque, em geral, a utilização de cada uma das tecnologias empregadas naquelas abordagens demanda uma quantidade de recursos computacionais superior à requerida pelo modelo tradicional. Além disso, uma nova classe de equipamentos aptos a conectarem-se a redes TCP/IP tem sido introduzida, tais como telefones celulares e PDAs. Esses equipamentos ainda oferecem, em maior ou menor escala, restrições à aplicação daquelas tecnologias.

Conforme foi visto no Capítulo 4, uma das considerações gerais relativas à utilização de JINI na construção de NMSs diz respeito à sua aplicabilidade em dispositivos com baixo poder computacional. Essa propriedade não é exclusiva de JINI. Em particular, as outras duas tecnologias alternativas consideradas também permitem a utilização de componentes de software em dispositivos daquela categoria.

A especificação *Minimum CORBA* [OMG, 2002a], por exemplo, define um conjunto mínimo de características CORBA para aplicação em dispositivos com baixo poder computacional, sem comprometimento de portabilidade e interoperabilidade com aplicações CORBA instaladas em dispositivos de outras categorias.

A manutenção da portabilidade e interoperabilidade (características fundamentais da tecnologia CORBA) é obtida com o suporte completo da OMG IDL. Desta forma, as mesmas aplicações que são executadas em dispositivos que suportam a especificação CORBA completa podem ser executadas em dispositivos suportando *Minimum* CORBA, satisfeitos os requisitos mínimos relacionados aos recursos necessários, naturalmente.

No entanto, haja vista ser projetada especificamente para a construção de sistemas embarcados (cuja funcionalidade é quase sempre conhecida em tempo de compilação), *Minimum* CORBA não inclui componentes relacionados aos aspectos dinâmicos de CORBA, como a DSI por exemplo. Portanto, a fim de efetuarem invocações remotas dinamicamente, tais aplicações precisam elas próprias suprirem esta funcionalidade. Isso torna quase inviável a construção de aplicações com tal funcionalidade. Dentre os produtos que suportam esta especificação estão o *OpenFusion* $E^{*}ORB$, da PrismTech Ltd., e *Gibraltar*, da Bionic Buffalo Corporation.

A utilização de Aplicações de Gerência executando a partir de dispositivos com baixo poder computacional e comunicando-se com Web Services impõe alguns requisitos. Dentre eles está a presença, nesses dispositivos, de alguma espécie de *parser* XML capaz de codificar e decodificar mensagens em XML-RPC ou SOAP. Outro requisito é a capacidade de enviar e receber tais mensagens. Essas funcionalidades estão presentes nos *stubs* para os Web Services.

Para que a invocação remota de métodos junto a um Web Service possa ser realizada dinamicamente, dois passos devem ser seguidos. O primeiro deles é a obtenção, junto a um Registro UDDI, de um arquivo WSDL referente à interface implementada pelo Web Service. O segundo passo é a geração dinâmica do seu *stub* correspondente, a partir daquele arquivo WSDL. Essa funcionalidade naturalmente elevaria o consumo de recursos computacionais no dispositivo onde está instalada a Aplicação de Gerência.

Produtos que utilizam a tecnologia JINI embarcada em dispositivos com baixo poder computacional são desenvolvidos e disponibilizados por algumas empresas. A canadense PsiNaptic Inc., por

exemplo, é responsável pelo desenvolvimento do *JMatos*, um LUS escrito em Java e que possui um consumo de memória inferior a 100 KBytes, sendo portanto adequada a sua utilização em dispositivos com baixo poder computacional. Esse produto passou pelos testes de compatibilidade contidos na versão 1.2b do TCK (*Technology Compatibility Kit*) disponibilizado pela Sun Microsystems, tendo sido utilizado em vários dispositivos equipados com processadores, sistemas operacionais e JVMs diferentes.

Um outro produto da PsiNaptic, denominado *CMatos*, é funcionalmente equivalente ao JMatos. Porém, por ser desenvolvida utilizando C, e não Java, esta implementação possui um consumo de memória inferior a 60 KBytes. Implementado em um número de microprocessadores, que inclui o *Dallas Semiconductor DS80C400 Network Controller* e o *Cambrige Silicon Radio BlueCore2*, o CMatos estende JINI a dispositivos que não suportam a tecnologia Java. Esses dispositivos podem oferecer Serviços que vão desde um simples *driver* até uma aplicação completa possuindo uma interface gráfica.

Apesar dessas iniciativas, para utilização de Aplicações de Gerência e Serviços JINI associados a dispositivos com baixo poder computacional recomenda-se o emprego do Projeto *Surrogate*, conforme descrito nesta Tese.

A Figura 7.5 ilustra as possibilidades vislumbradas em cada tecnologia no tocante à utilização de Aplicações de Gerência a partir de dispositivos com baixo poder computacional.

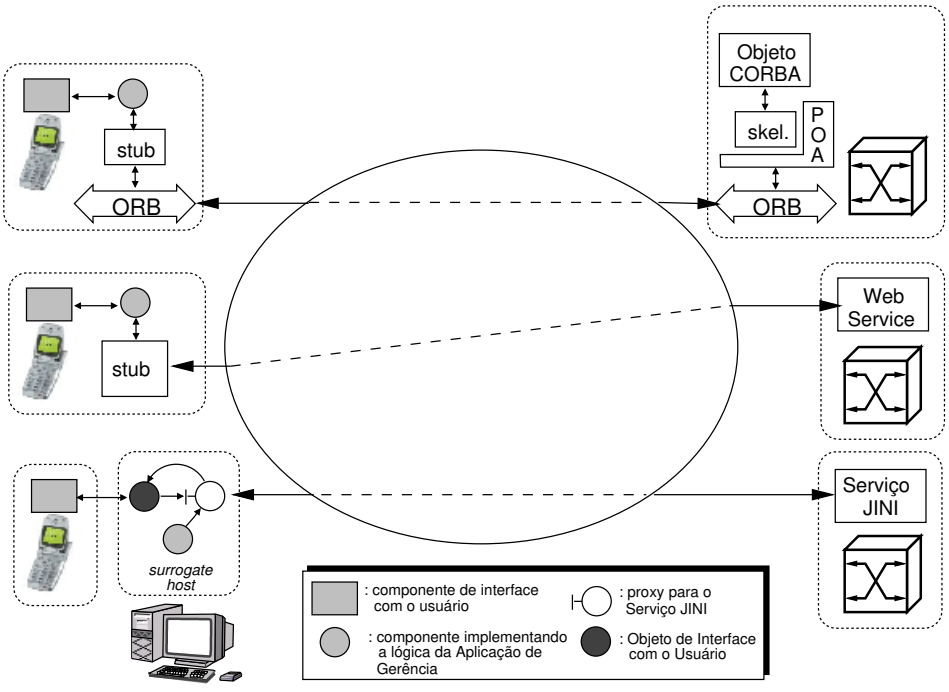

Figura 7.5: Aplicações de Gerência executando em dispositivos com baixo poder computacional.

Nesse quesito, afirma-se que, embora requeira a presença de um *surrogate host* em execução numa plataforma adequada, a solução baseada em JINI possui viabilidade superior às soluções

análogas descritas para CORBA e Web Services nesta subseção.

Essa superioridade vem do fato de que a utilização do Projeto *Surrogate* possibilita a introdução de tais dispositivos em NMSs sem que haja um real comprometimento da funcionalidade oferecida pelas suas respectivas Aplicações de Gerência, contrariamente ao que ocorre na maioria das vezes para componentes que utilizam ORBs implementando a especificação *Minimum* CORBA. Isso ocorre porque, na Arquitetura *Surrogate*, os *surrogates* relativos às Aplicações de Gerência são executados por *surrogate hosts* em plataformas nas quais há recursos computacionais suficientes para o descarregamento dinâmico de código, uma das características chave de JINI.

Além disso, a presença desses *surrogate hosts* permite que os dispositivos hospedem apenas um componente de software capaz de interfacear com o usuário. As abordagens baseadas na utilização de CORBA e Web Services não possuem essa propriedade. Nessas abordagens, todos os componentes de uma Aplicação de Gerência estão presentes no dispositivo, desde aqueles que implementam a lógica das atividades de gerência até os que compõem a Interface com o Usuário.

Naturalmente, as diferenças conceituais apontadas nos parágrafos anteriores desaparecem quando considera-se o caso de os dispositivos desempenharem o papel de sistemas gerenciados. Isso ocorre porque estes dispositivos já detêm todo o código necessário para processar e atender às solicitações emitidas pelas Aplicações de Gerência.

7.4.5 Análise Comparativa

Algumas informações referentes às tecnologias empregadas para gerência de redes TCP/IP citadas ao longo desta Tese, e mais especificamente neste capítulo, encontram-se sumarizadas na Tabela 7.4.

Conforme descrito na Subseção 7.4.1, cada uma das tecnologias alternativas investigadas possui sua própria maneira de definir os seus Serviços. Essas maneiras, por sua vez, possuem relação estreita com as possibilidades de implementação daqueles Serviços.

Por exemplo, haja vista a definição de interfaces para Objetos CORBA e Web Services ser realizada utilizando-se as linguagens declarativas OMG IDL e WSDL, tem-se para aqueles Serviços uma multiplicidade de linguagens de programação que podem ser utilizadas para desenvolvê-los. Serviços JINI, por outro lado, requerem tipicamente a utilização de Java para seu desenvolvimento[4], pois suas interfaces são definidas utilizando-se esta linguagem.

Por outro lado, observa-se no trecho do arquivo WSDL ilustrado na Subseção 7.4.1, a presença de parâmetros que identificam mais do que a funcionalidade oferecida pelo Web Service. Mais especificamente, o elemento `wsdlsoap:binding` identifica a forma segundo a qual as mensagens codificadas em SOAP são transportadas (no caso, utilizando-se o HTTP). O elemento `service`, por sua vez, identifica a localização do Web Service (no caso, a URL `http://143.106.50.214:8080/axis/services/WorkStationServer`).

[4]Pelo menos no desenvolvimento do *proxy* utilizado pelo Cliente, além dos componentes responsáveis pelos processos de descoberta e adesão.

Item	SNMP/SMI	CORBA	Web Services	JINI
definição de interfaces	ASN.1	OMG IDL	WSDL	Interfaces Java
protocolo utilizado nas comunicações	SNMP	GIOP implementado sobre TCP/IP (IIOP, ...)	SOAP mapeado em HTTP, SMTP, ...	qualquer protocolo, virtualmente
linguagens utilizadas para implementação	qualquer uma, virtualmente	qualquer uma, virtualmente	qualquer uma, virtualmente	Java
registro/consulta de Serviços	não possui	Serviços de Nomeação e de *Trading*	UDDI	LUS
suporte ao uso de eventos	PDUs *Trap* e *Inform*	Serviços de Evento e Notificação	especificações WSN	intrínseco à tecnologia
suporte ao uso de segurança	USM/VACM	CORBAsec	WSS, XMLDS, Encriptação XML, SAML,...	à cargo do protocolo utilizado
suporte ao uso de transações	não possui	OTS	especificações WST	intrínseco à tecnologia
suporte ao uso de dispositivos com restrições computacionais	intrínseco à tecnologia	*Minimum* CORBA	não possui nenhum suporte padrão, apesar de ser possível	Projeto *Surrogate*
suporte ao uso de Interfaces com o Usuário flexíveis	não possui	Interceptadores Portáteis	não possui	Projeto *ServiceUI*

Tabela 7.4: Quadro comparativo entre as tecnologias.

Do parágrafo anterior, pode-se concluir que a definição de um Web Service a partir de seu respectivo arquivo WSDL oferece menos possibilidades arquiteturais de implementação aos seus desenvolvedores, quando comparada à definição da interface implementada por um Serviço JINI. Mesmo permitindo, à semelhança de CORBA, que várias linguagens de programação possam ser utilizadas na implementação daquele Web Service. Ademais, contrariamente à definição de interfaces implementadas por Objetos CORBA e Serviços JINI, a definição do Web Service não abstrai a sua localização.

Em [Sloten et al., 2004], são apontadas algumas tentativas de modularizar esta definição. Essas tentativas compreendem a utilização dos elementos $< import >$ e $< include >$, permitindo agregar componentes externos a uma descrição WSDL, e a adoção da recomendação emitida pelo Comitê Técnico UDDI, que consiste em separar a parte da "definição da interface do Serviço" (os elementos

$< types >$, $< porttype >$ e $< binding >$) da parte da "definição da implementação do Serviço" (o elemento $< service >$). Sobre esse último item, [Sloten et al., 2004] faz a ressalva de que, na verdade, o elemento $< binding >$ também deve ser separado da parte da "definição da interface do Serviço". Independentemente dessa modularização, os comentários efetuados no parágrafo anterior permanecem válidos, haja vista que o Web Service é definido a partir da junção dessas duas "partes", mesmo estando elas definidas em arquivos separados.

A fim de realizar uma comparação justa entre os protótipos desenvolvidos na Subseção 7.4.1, todos eles foram implementados utilizando uma mesma linguagem: Java. Mesmo assim, os resultados obtidos a partir daqueles protótipos devem ser analisados com cuidado. Isso porque, similarmente ao comentado em [Pras et al., 2004], os valores obtidos para os tempos de invocação, ilustrados pela Figura 7.3 e sumarizados na Tabela 7.2, são altamente dependentes das plataformas de hardware/software utilizadas e de especificidades das implementações.

Essas especificidades referem-se tanto às APIs utilizadas quanto à qualidade das implementações que as utilizam. Portanto, tais valores podem servir apenas de indicações para possíveis comparações de desempenho, e não devem ser tomadas como resultados definitivos.

Por outro lado, os valores obtidos para os tráfegos gerados, por dependerem das especificações dos protocolos utilizados em cada um dos protótipos, devem ser os mesmos a serem obtidos por outros protótipos construídos.

É também válido ressaltar que as comparações realizadas entre os protótipos desenvolvidos constituem-se, na verdade, em comparações entre os protocolos utilizados por eles. Porém, Web Services, por exemplo, é uma tecnologia que permite que outros protocolos além do HTTP sejam utilizados para o transporte de mensagens SOAP. Essa flexibilidade é ainda maior para JINI, que, conforme apontado na Tabela 7.4, pode utilizar virtualmente qualquer protocolo.

Na verdade, se tecnologias como SNMP/SMI, CORBA e Web Service assumem (em maior ou menor grau) um compromisso quanto ao protocolo utilizado na comunicação entre seus respectivos componentes arquiteturais, JINI não possui qualquer compromisso no que diz respeito a isso. Conforme foi descrito ao longo desta Tese, o único compromisso assumido nesta tecnologia refere-se à interface implementada pelos Serviços nela definidos.

Com relação à infra-estrutura provida, foi visto que todas as tecnologias investigadas provêem suporte, em graus variados, ao registro e descoberta de Serviços e à utilização de eventos/notificações, segurança e transações. A utilização da infra-estrutura provida por CORBA é incentivada nas abordagens descritas em [Pavlou, 1997], [Pavlou, 1999] e, mais enfaticamente, em [Pavon, 1999].

Por outro lado, as abordagens baseadas em Web Services descritas concentram-se principalmente na modelagem de operações ([Schönwälder et al., 2003] e [Sloten et al., 2004]) e na investigação da eficiência de seus mecanismos de comunicação ([Drevers et al., 2004] e [Pras et al., 2004]), se possível comparando-os com os mecanismos presentes em outras abordagens ([Pras et al., 2004]).

De fato, nas abordagens baseadas em Web Services pouca atenção é dada na utilização da infra-estrutura relacionada àquela tecnologia. Isso pode ser explicado levando-se em conta que alguns dos mecanismos que compõem esta infra-estrutura possuem um nível de amadurecimento inferior aos dos mecanismos análogos definidos para CORBA. Na verdade, boa parte deles sequer existia na época da introdução das abordagens.

Com relação à JINI, a proposta descrita nesta Tese recomenda, mesmo que indiretamente, a adoção da infra-estrutura associada àquela tecnologia. Das iniciativas relacionadas à utilização de LUSs responsáveis por Domínios Gerenciados ao emprego dos mecanismos de segurança intercambiáveis, passando pela utilização das funcionalidades de geração e manipulação de eventos, atenção particular é dada a este item como forma de aumentar a produtividade no desenvolvimento de NMSs. A utilização dessa infra-estrutura é direta, haja vista que ela encontra-se totalmente integrada nas versões do JTSK disponibilizadas.

Quando é analisada a possibilidade de emprego de Interfaces com o Usuário flexíveis em cada uma das tecnologias alternativas (e mesmo no modelo tradicional), foi visto que essa é uma característica oferecida apenas por CORBA e pela tecnologia na qual baseia-se a abordagem proposta.

No que diz respeito à aplicabilidade em dispositivos com baixo poder computacional, também foi visto que, em maior ou menor grau, todas as tecnologias possuem mecanismos que buscam viabilizar essa característica. Tais mecanismos incluem desde a transferência da funcionalidade para dispositivos com maior disponibilidade de recursos computacionais, como é o caso de JINI com a adoção do Projeto *Surrogate*, até a redução da funcionalidade provida pela tecnologia, como é o caso da especificação *Minimum* CORBA. No caso de Web Services, como foi visto, a falta de um mecanismo auxiliar específico àquela tarefa é decorrente de exigências relativamente pequenas para a sua realização.

Conforme já foi comentado, comparações entre JINI e as demais tecnologias, nos moldes do que foi feito neste capítulo e em outras referências, devem ser analisadas com cuidado. Particularmente, quando essas comparações incluem medidas de desempenho entre os protocolos utilizados naquelas tecnologias.

Isso ocorre porque algumas tecnologias podem utilizar vários daqueles protocolos. No caso de JINI, especificamente, é possível a utilização de todos os protocolos passíveis de serem utilizados nas demais tecnologias investigadas. Mais ainda, JINI permite também o emprego daquelas próprias tecnologias.

No caso ilustrado pela Figura 7.6, por exemplo, é necessário apenas que componentes de software escritos em Java (não mostrados na Figura) participem dos protocolos de descoberta e adesão e registrem *proxies* serializados. Esses *proxies*, em última instância, comunicam-se com aplicações associadas aos dispositivos.

Também é ilustrado na Figura 7.6 que é possível a cooperação entre JINI e a abordagem tradicional baseada no binômio SNMP/SMI. Essa cooperação já foi demonstrada nesta Tese, particularmente nos capítulos referentes às provas de conceitos construídas para validação da proposta introduzida. Conforme foi visto naqueles capítulos, seus respectivos NMSs utilizam SNMPv3 para a coleta de informações e reforço de Regras de Política em dispositivos. Constituem-se, portanto, provas incontestáveis de que a tecnologia JINI pode ser utilizada para levar a sistemas legados as tão almejadas características de flexibilidade e automação em suas operações.

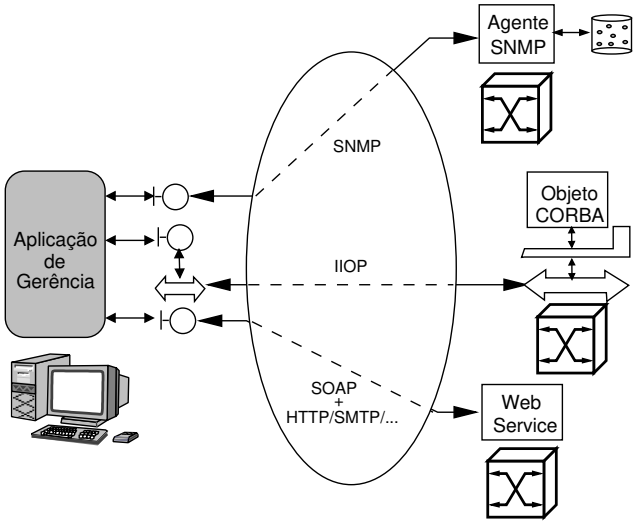

Figura 7.6: Integração entre JINI e as demais tecnologias.

7.5 Conclusão

Este capítulo realizou uma comparação entre JINI e outras tecnologias popularmente utilizadas para a gerência de redes. Mais especificamente, JINI foi comparada com a tecnologia utilizada no modelo tradicional e com outras duas tecnologias utilizadas na criação de abordagens revolucionárias: CORBA e Web Services.

Após uma breve descrição das deficiências observadas no modelo tradicional, procedeu-se a apresentação acerca de CORBA e Web Services, apontando algumas abordagens nela baseadas. Em seguida, as tecnologias foram confrontadas levando-se em consideração aspectos relacionados à modelagem de Serviços, utilização da infra-estrutura, possibilidade de emprego de Interfaces com o Usuário flexíveis e aplicabilidade em dispositivos com baixo poder computacional.

Esse confronto culminou com a constatação de que JINI é uma tecnologia que pode tranqüilamente ser posta ao lado das demais tecnologias investigadas com relação à construção de NMSs, por vezes apresentando até mesmo vantagens notáveis frente a elas.

A despeito de sua forte dependência em relação à Java, JINI conta com uma infra-estrutura forte e madura, capaz de atender aos requisitos impostos ao desenvolvimento de NMSs e prover-lhes ainda características relacionadas à flexibilidade e automação em suas operações. Além disso, essa tecnologia possui um mecanismo para o emprego de Interfaces com o Usuário flexíveis, além de um suporte adequado ao emprego de dispositivos com baixo poder computacional naqueles sistemas.

Com efeito, pode-se dizer que a abordagem revolucionária descrita nesta Tese, por ser baseada na utilização dessa tecnologia, permite por definição a utilização de quaisquer protocolos empregados nas demais tecnologias descritas até então. Mais ainda, pode-se afirmar que esta abordagem vislumbra, inclusive, a utilização daquelas próprias tecnologias na construção de NMSs.

Capítulo 8

Considerações finais

8.1 Introdução

Esta Tese apresentou um estudo da aplicabilidade da tecnologia JINI na construção de NMSs, isto é, como JINI pode ser utilizada de maneira eficaz no projeto de NMSs com vistas a permitir que esses sistemas usufruam dos benefícios oferecidos por aquela tecnologia.

Conforme visto ao longo da Tese, este estudo resultou em uma proposta que deu origem a uma série de considerações gerais. De forma a validar essas considerações, duas provas de conceitos foram desenvolvidas. Em seguida, JINI foi comparada com outras tecnologias levando-se em conta aspectos relevantes a construção de NMSs. Dessa comparação, conclui-se que JINI possui viabilidade equivalente (e às vezes superior) às demais tecnologias consideradas.

Neste capítulo, são feitas as considerações finais acerca desse estudo. Para tal, além desta seção, este Capítulo compreende outras duas: na Seção 8.2, são descritos os resultados obtidos a partir desta Tese, incluindo-se aí resultados de âmbito geral. Em seguida, a Seção 8.3 aborda alguns possíveis desdobramentos que podem surgir a partir dela, fomentando o surgimento de trabalhos futuros.

8.2 Objetivos alcançados

Inegavelmente, a contribuição fundamental desta Tese é a conclusão de que JINI é uma tecnologia que, à semelhança de outras tecnologias de âmbito geral, mostra-se adequada à construção de aplicações voltadas à gerência de redes. Essa conclusão faz com que a proposta introduzida no Capítulo 4 situe-se no mesmo patamar das propostas baseadas em CORBA e Web Services descritas no capítulo anterior.

Em particular, a proposta introduzida nesta Tese destaca-se frente às demais abordagens revolucionárias descritas pelos seguintes motivos:

1. ela incentiva fortemente a utilização da infra-estrutura relacionada à JINI (registro/descoberta de Serviços, geração/manipulação de eventos, segurança e transações);

2. ela aplica um mecanismo dessa tecnologia (o Projeto *ServiceUI*) para a utilização de Interfaces com o Usuário flexíveis, uma funcionalidade antes obtível somente a partir de Interceptadores Portáteis CORBA;

3. ela realça a possibilidade de inclusão de dispositivos possuindo níveis distintos de poder computacional, também mediante utilização de um mecanismo específico a tal tarefa (o Projeto *Surrogate*).

Além de promover a utilização da tecnologia JINI especificamente na gerência de redes TCP/IP, esta Tese buscou desmistificar o seu papel na construção de sistemas distribuídos como um todo.

Desde o seu surgimento, no final da década de 90, JINI vem sendo considerada uma tecnologia que, baseado no conceito de computação ubíqua, busca prover conectividade a todo e qualquer tipo de dispositivo. Atenção foi dada inicialmente àqueles que, possuindo restrições computacionais, dedicam-se a uma determinada tarefa. É o caso de dispositivos domésticos e seus respectivos sistemas embarcados, por exemplo.

A manutenção desse foco por parte da maioria das pessoas tem levado a uma má compreensão das reais funcionalidades oferecidas por JINI. Web Services, por exemplo, vem sendo considerada como a tecnologia responsável pela divulgação da SOA (*Services Oriented Architecture*). Em termos práticos, SOA consiste num estilo arquitetural cujo objetivo é conseguir-se um acoplamento fraco entre os componentes participantes de um sistema.

É válido ressaltar que os NMSs baseados em JINI descritos nos capítulos 5 e 6 possuem tal propriedade. Falhas na promoção dessa tecnologia, todavia, fazem com que ela se torne cada vez mais marginalizada com relação a Web Services. Essa é uma das raras circunstâncias na qual o sucesso de mercado nada tem a ver com mérito técnico.

Para a construção de NMSs, que foi o escopo desta Tese, JINI apresentou-se também como uma tecnologia viável, em alguns aspectos até mesmo superando Web Service. De fato, a flexibilidade e automação obtidas por esses sistemas com a adoção de JINI em seu projeto são difíceis de serem obtidas com a utilização de Web Services.

8.3 Direções futuras

Um conjunto de possibilidades descortinam-se a partir do estudo descrito nesta Tese. Essas possibilidades podem ser enquadradas no desenvolvimento de NMSs com funcionalidades específicas e na integração de outras tecnologias na proposta introduzida.

Diversos protótipos de NMSs têm sido desenvolvidos utilizando CORBA e (mais recentemente) Web Services, validando cada vez mais as abordagens baseadas naquelas tecnologias. Da mesma forma, vários outros NMSs baseados em JINI (além daqueles descritos nos capítulos 5 e 6) podem ser construídos com o mesmo objetivo. Afinal de contas, desde a adoção do SNMP sabe-se que a disponibilidade de aplicações é um fator de peso para a aceitação de uma tecnologia.

Esses novos NMSs baseados em JINI podem ser projetados de forma a endereçar necessidades específicas às tecnologias de comunicação ou aplicações vigentes. Seu desenvolvimento pode, inclusive, dar origem a novas considerações que se somariam as descritas nesta Tese.

Tais NMSs podem empregar, em seu projeto, níveis de granularidade menores que aqueles empregados no projeto dos NMSs descritos nesta Tese. A presença de vários Serviços associados a um sistema gerenciado pode requerer um nível de orquestração entre esses Serviços. Modelos colaborativos e hierárquicos entre eles podem ser considerados.

Outra sugestão para trabalhos futuros é a integração entre JINI e outras tecnologias no desenvolvimento de NMSs. Uma sugestão neste sentido é a construção de *gateways* SNMP/JINI, nos moldes do que é feito em [Neisse et al., 2004]. Um destes *gateways* poderia ser definido utilizando-se o menor nível de granularidade possível, possuindo apenas um Serviço cujos métodos seriam análogos às PDUs SNMP. O outro utilizaria o maior nível de granularidade, compreendendo Serviços modelando MOs escalares e tabulares de uma MIB ou de parte dela. A idéia por trás dessa sugestão é a medição do tempo de resposta, consumo de memória e tráfego gerado em cada um desses protótipos com vistas a avaliar de que forma estes *gateways* podem ser melhor projetados.

Além da integração entre JINI e o binômio SNMP/SMI, encoraja-se também a sua combinação com as tecnologias utilizadas nas outras abordagens revolucionárias. Também nesse contexto, algumas iniciativas pré-existentes podem ser consideradas.

O Projeto *Judy* [Asberry, 2005], por exemplo, busca integrar JINI e Web Services, de forma que Serviços JINI possam ser utilizados por Clientes Web Services e vice-versa. Apesar daquele Projeto não ter até então resultado em nenhuma especificação, esforços no sentido de produzí-las e aplicá-las na área de gerência de redes TCP/IP devem ser estimulados.

Encoraja-se também o estudo acerca de mecanismos que garantam a interoperabilidade entre JINI e *Agentes Móveis* [Karmouch e Pham, 1998]. Estudos no sentido de aplicar aquele paradigma na construção de NMSs têm sido conduzidos há algum tempo. Em [Bieszczad et al., 1998], por exemplo, é investigada a utilização de Agentes Móveis na construção de NMSs endereçando as cinco áreas funcionais definidas para a gerência OSI. Aplicações em três destas áreas (mais especificamente as áreas de gerência de falhas, configuração e desempenho), são descritas em [White et al., 1998], [Pagurek et al., 1998] e [Bohoris et al., 2000], respectivamente. Por fim, em [Pagurek et al., 2000] é investigada a integração esta abordagem com o modelo tradicional.

Algumas iniciativas têm surgido em direção ao desenvolvimento de Agentes Móveis utilizando JINI, tais como *JIMA*, *Ronin* e, mais recentemente, *Agent*. JIMA [Kilander et al., 1999] é uma infra-estrutura baseada em Java e JINI para documentos ativos na forma de Agentes Móveis. Ronin [Chen, 1999] é um modelo que provê extensões aos atributos que descrevem Serviços JINI em LUSs. Agents [Chen et al., 2005] é um Projeto surgido no âmbito da Comunidade JINI. Embora ainda não experimentado nenhum progresso significante, espera-se que esse Projeto possa tirar

proveito dos mecanismos de segurança atualmente disponíveis em JINI, de forma a utilizá-los em NMSs baseados em Agentes Móveis.

A presente seção finaliza esta Tese, cuja contribuição inegável foi propor e investigar o uso da tecnologia JINI na construção de NMSs, à semelhança do que foi feito no passado com tecnologias como CORBA e Web Services, por exemplo. Conforme visto nesta seção, também à semelhança daquelas iniciativas, esta Tese abre portas para novas possibilidades. Que surjam, então.

Referências Bibliográficas

[Adams, 1996] Adams, C. (1996). *The Simple Public-Key GSS-API Mechanism (SPKM) - rfc 2025*.

[AdventNet, 2005] AdventNet, I. (2005). Adventnet snmp api: Java snmp api/snmp stack/snmp library for snmp management. http://snmp.adventnet.com/.

[Andrews et al., 2003] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., e Weerawarana, S. (2003). *Business Process Execution Language for Web Services - version 1,1*. BEA Systems, IBM Inc., Microsoft Corp., SAP AG e Siebel Systems.

[ANSI, 1983] ANSI (1983). *ANSI X3.106: American National Standard for Information Systems - Data Link Encryption*. American National Standards Institute.

[Asberry, 2005] Asberry, D. (2005). Página web do projeto judy na comunidade jini. http://judy.jini.org.

[Bartel et al., 2002] Bartel, M., Boyer, J., Fox, B., e LaMacchia, B. (2002). *XML Signature Syntax and Processing (W3C Recommendation)*. World Wide Web Consortium.

[Bieszczad et al., 1998] Bieszczad, A., Pagurek, B., e White, T. (1998). Mobile agents for network management. *IEEE Communications Surveys*, 1(1):2–9.

[Blumenthal e Wijnen, 1998] Blumenthal, U. e Wijnen, B. (1998). *User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3) - rfc 2274*.

[Bohoris et al., 2000] Bohoris, C., Pavlou, G., e Cruickshank, H. (2000). Using mobile agents for network performance management. In *Proceedings of the 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS'00)*, páginas 637–652. Honolulu, Havaí, EUA.

[Booth et al., 2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., e Orchard, D. (2004). *Web Services Architecture*. World Wide Web Consortium (W3C).

[Bray et al., 1998] Bray, T., Paoli, J., e Sperberg-McQueen, C. M. (1998). *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium (W3C).

[Bruins, 1996] Bruins, B. (1996). Some experiences with emerging management technologies. *The Simple Times - The Quarterly Newsletter of SNMP Technology, Comment, and Events*, 4(3).

[Cabrera et al., 2005a] Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Shewchuck, J., e Storey, T. (2005a). *Web Services Coordination (WS-Coordination)*. Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machine Corporation, IONA Technologies e Microsoft Corporation, Inc.

[Cabrera et al., 2005b] Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Storey, T., e Thatte, S. (2005b). *Web Services Atomic Transaction (WS-Atomic Transaction)*. Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machine Corporation, IONA Technologies e Microsoft Corporation, Inc.

[Cabrera et al., 2005c] Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Joyce, S., Klein, J., Langworthy, D., Little, M., Leymann, F., Newcomer, E., Orchard, D., Robinson, I., Storey, T., e Thatte, S. (2005c). *Web Services Business Activity Framework (WS-BusinessActivity)*. Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machine Corporation, IONA Technologies e Microsoft Corporation, Inc.

[Cantor et al., 2005] Cantor, S., Kemp, J., Philpott, R., e Maler, E. (2005). *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0*. Organization for the Advancement of Structured Information Standards.

[Case et al., 1990] Case, J., Fedor, M., Schoffstall, M., e Davin, J. (1990). *A Simple Network Management Protocol (SNMP) - rfc 1157*.

[Case et al., 1996] Case, J., McCloghrie, K., Rose, M., e Waldbusser, S. (1996). *Introduction to Community-based SNMPv2 - rfc 1901*.

[Case et al., 1993] Case, J. D., McCloghrie, K., Rose, M. T., e Waldbusser, S. (1993). *Manager-to-Manager Management Information Base - rfc 1451*.

[Chan et al., 2001] Chan, K. H., Seligson, J., Gai, S., McCloghrie, K., Herzog, S., Reichmeyer, F., Yavatkar, R., e Smith, A. (2001). *COPS Usage for Policy Provisioning (COPS-PR) - rfc 3084*.

[Chapman e Montesi, 1995] Chapman, M. e Montesi, S. (1995). *Overall Concepts and Principles of TINA*. Telecommunications Information Networking Architecture Consortium.

[Chappell e Liu, 2005] Chappell, D. e Liu, L. (2005). *Web Services Brokered Notification 1.3 (WS-BrokeredNotification)*. Organization for the Advancement of Structured Information Standards.

[Chen, 1999] Chen, H. (1999). Developing a dynamic distributed intelligent agent framework based on the jini architecture. Dissertação de Mestrado, University of Maryland Baltimore County. Baltimore, Maryland, EUA.

[Chen et al., 2005] Chen, H., Wong, H., e Saltmarsh, C. (2005). Página web do projeto agents na comunidade jini. http://agents.jini.org.

[Chinnici et al., 2005] Chinnici, R., Moreau, J.-J., Ryman, A., e Weerawarana, S. (2005). *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. World Wide Web Consortium (W3C).

[Chisholm e Romascanu, 2004] Chisholm, S. e Romascanu, D. (2004). *Alarm Management Information Base (MIB) - rfc3877*.

[Choi et al., 2003] Choi, M.-J., Hong, J. W., e Ju, H.-T. (2003). Xml-based network management for ip networks. *Eletronics and Telecommunications Research Institute (ETRI) Journal*, 25(6):445–463.

[Clement et al., 2004] Clement, L., Hately, A., Riegen, C. v., e Rogers, T. (2004). *UDDI Version 3.0.2*. Organization for the Advancement of Structured Information Standards.

[Davin et al., 1987] Davin, J., Case, J., Fedor, M., e Schoffstall, M. (1987). *A Simple Gateway Monitoring Protocol - rfc 1028*.

[Dierks e Allen, 1999] Dierks, T. e Allen, C. (1999). *The TLS Protocol Version 1.0 - rfc 2246*.

[DMTF, 2005] DMTF (2005). Página web contendo as especificações e schemas cim. http://www.dmtf.org/standards/cim/.

[Drevers et al., 2004] Drevers, T., Meent, R. v., e Pras, A. (2004). Prototyping web services based network monitoring. In *Proceedings of the 10th EUNICE Summer School (EUNICE'04) and IFIP WG 6.3 Workshop*, páginas 135–142. Tampere, Finlândia.

[Durham et al., 2000] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R., e Sastry, A. (2000). *The COPS (Common Open Policy Service) Protocol*.

[Ethereal, Inc., 2005] Ethereal, Inc. (2005). Etheral: A network protocol analyzer.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., e Berners-Lee, T. (1999). *Hypertext Transfer Protocol – HTTP/1.1*.

[Gemplus, 2002] Gemplus (2002). *Smart card Interconnect Specification - Gemplus proposal, Version 0.2*.

[Goldszmidt e Yemini, 1995] Goldszmidt, G. e Yemini, Y. (1995). Distributed management by delegation. In *Proceedings of the 15th IEEE International Conference on Distributed Computing (ICDC'95)*, páginas 333–341. Vancouver, Colúmbia Britânica, Canadá.

[Good, 2000] Good, G. (2000). *The LDAP Data Interchange Format (LDIF) - Technical Specification - rfc 2849*.

[Graham et al., 2005] Graham, S., Hull, D., e Murray, B. (2005). *Web Services Base Notification 1.3 (WS-BaseNotification)*. Organization for the Advancement of Structured Information Standards.

[Hallam-Baker e MySore, 2005] Hallam-Baker, P. e MySore, S. H. (2005). *XML Key Management Specification (XKMS 2.0)*. World Wide Web Consortium.

[Hardaker, 2005] Hardaker, W. (2005). Mib ucd-snmp. Disponível em http://net-snmp.sourceforge.net/docs/mibs/UCD-SNMP-MIB.txt.

[Hardaker et al., 2005] Hardaker, W., Shield, D., Story, R., Baer, M., Marzot, G. S., Baggesen, N., e Slifcak, M. J. (2005). Página web do projeto net-snmp. http://net-snmp.sourceforge.net/.

[Harrington et al., 1998] Harrington, D., Presuhn, R., e Wijnen, B. (1998). *An Architecture for Describing SNMP Management Frameworks - rfc 2271*.

[IETF, 2002a] IETF (2002a). Evolution of snmp (eos) charter - página web do eos wg. http://www.ietf.org/html.charters/OLD/eos-charter.html.

[IETF, 2002b] IETF (2002b). Next generation structure of management information (sming) charter - página web do sming wg. http://www.ietf.org/html.charters/OLD/sming-charter.html.

[IETF, 2004] IETF (2004). Resource allocation protocol (rap) charter - página web do rap. http://www.ietf.org/html.charters/rap-charter.html.

[IETF, 2005a] IETF (2005a). Distributed management (disman) charter - página web do disman. http://www.ietf.org/html.charters/disman-charter.html.

[IETF, 2005b] IETF (2005b). Remote network monitoring (rmonmib) charter - página web do rmon wg. http://www.ietf.org/html.charters/rmonmib-charter.html.

[Imamura et al., 2002] Imamura, T., Dillaway, B., e Simon, E. (2002). *XML Encryption Syntax and Processing (W3C Recommendation)*. World Wide Web Consortium.

[ISO/IEC e ITU-T, 1991] ISO/IEC e ITU-T (1991). *ISO 10040 and ITU Recommendation X.701: Information Processing Systems – Open Systems Interconnection – Systems Management Overview*.

[ISO/IEC e ITU-T, 1992] ISO/IEC e ITU-T (1992). *ISO/IEC 10165-4 and ITU Recommendation X.722: Information technology – Open Systems Interconnection, Structure of Management Information – Part 4: Guidelines for Definition of Managed Objects*.

[ISO/IEC e ITU-T, 1993] ISO/IEC e ITU-T (1993). *ISO/IEC 10746-1 and ITU-T Recommendation X.901: Information Technology – Open Distributed Processing – Reference Model: Overview*.

[ISO/IEC e ITU-T, 1998] ISO/IEC e ITU-T (1998). *ISO/IEC 9596 and ITU Recommendation X.711: Information technology – Open Systems Interconnection – Common management information protocol – Part 1: Specification*.

[ISO/IEC e ITU-T, 1999] ISO/IEC e ITU-T (1999). *ISO/IEC 14750 and ITU-T Recommendation X.920: Information technology – Open Distributed Processing – Interface Definition Language*.

[ITU-T, 1992] ITU-T (1992). *ITU-T Recommendation M.3010: Principles for a Telecommunications Management Network (TMN)*. The ITU Telecommunication Standardization Sector, Study Group IV.

[Karmouch e Pham, 1998] Karmouch, A. e Pham, V. A. (1998). Mobile software agents: An overview. *IEEE Communications Magazine*, 10(10):26–37.

[Kavasseri e Stewart, 2000a] Kavasseri, R. e Stewart, B. (2000a). *Distributed Management Expression MIB - rfc2982*.

[Kavasseri e Stewart, 2000b] Kavasseri, R. e Stewart, B. (2000b). *Event MIB - rfc2981*.

[Kavasseri e Stewart, 2000c] Kavasseri, R. e Stewart, B. (2000c). *Notification Log MIB - rfc3014*.

[Kilander et al., 1999] Kilander, F., Werle, P., e Hansson, K. (1999). Jima: A jini-based infrastructure for active documents and mobile agents. In *Proceedings of the Personal Computing and Communication (PCC) Workshop*. Lund, Suécia.

[Kohl e Neuman, 1993] Kohl, J. e Neuman, B. C. (1993). *The Kerberos Network Authentication Service (V5) - rfc 1510*.

[Lam et al., 2004] Lam, H., Huynh, A., e Perkins, D. T. (2004). *Alarm Reporting Control Management Information Base (MIB) - rfc3878*.

[Landis e Vasudevan, 2002] Landis, S. e Vasudevan, V. (2002). Reaching out to the cell phone with jini. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS'2002)*, páginas 3821–3830. Havaí, EUA.

[Levi e Schöenwäelder, 2001] Levi, D. B. e Schöenwäelder, J. (2001). *Definition of Managed Objects for the Delegation of Management Scripts - rfc3165*.

[Levi e Schöenwäelder, 2002] Levi, D. B. e Schöenwäelder, J. (2002). *Definition of Managed Objects for Scheduling Management Operations - rfc3231*.

[McCloghrie e Rose, 1991] McCloghrie, K. e Rose, M. (1991). *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II - rfc1213*.

[Mitra, 2003] Mitra, N. (2003). *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium (W3C).

[Moore et al., 2001] Moore, B., Ellesson, E., Strassner, J., e Westerinen, A. (2001). *Policy Core Information Model – Version 1 Specification (rfc 3060)*.

[Moses, 2005] Moses, T. (2005). *eXtemsible Access Control Markup Language (XACML) Version 2.0*. Organization for the Advancement of Structured Information Standards.

[Nadalin et al., 2004] Nadalin, A., Kaler, C., Hallam-Baker, P., e Monzillo, R. (2004). *Web Services Security: SOAP Message Security 1.0 (OASIS Standard 200401)*. Organization for the Advancement of Structured Information Standards.

[Neisse et al., 2004] Neisse, R., Vianna, R. L., Granville, L. Z., Almeida, M. J. B., e Tarouco, L. M. R. (2004). Implementation and bandwidth consumption evaluation of snmp to web services gateway. In *Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS'04)*, páginas 715–728. Seul, Coréia do Sul.

[NIST, 1995] NIST (1995). *Secure Hash Algorithm (NIST FIPS 180-1)*. National Institute of Standards and Technology.

[Olstad et al., 1999] Olstad, L., Ramirez, J., Brady, C., e McHollan, B. (1999). Jini technology: Impromptu networking and its impact on telecommunications. In *Proceedings of Capstone 1999, University of Colorado at Boulder*. Colorado, EUA.

[OMG, 2000] OMG (2000). *Trading Object Service Specification (Version 1.0)*. Object Management Group, Inc.

[OMG, 2001] OMG (2001). *Interceptors Published Draft with CORBA 2.4+ Core Chapters*. Object Management Group, Inc.

[OMG, 2002a] OMG (2002a). *Minimum CORBA Specification*. Object Management Group, Inc.

[OMG, 2002b] OMG (2002b). *Security Service Specification (Version 1.8)*. Object Management Group, Inc.

[OMG, 2003] OMG (2003). *Transaction Service Specification (Version 1.4)*. Object Management Group, Inc.

[OMG, 2004a] OMG (2004a). *Common Object Request Broker Architecture: Core Specification (Version 3.0.3)*. Object Management Group, Inc.

[OMG, 2004b] OMG (2004b). *Event Service Specification (Version 1.2)*. Object Management Group, Inc.

[OMG, 2004c] OMG (2004c). *Naming Service Specification (Version 1.3)*. Object Management Group, Inc.

[OMG, 2004d] OMG (2004d). *Notification Service Specification (Version 1.1)*. Object Management Group, Inc.

[Pagurek et al., 1998] Pagurek, B., Li, Y., Bieszczad, A., e Susilo, G. (1998). Network configuration management in heterogeneous atm environments. In *Proceedings of the 3rd. International Workshop on Agents in Telecommunications Applications (IATA'98), AgentWorld'98*, páginas 72–88. Paris, França.

[Pagurek et al., 2000] Pagurek, B., Wang, Y., e White, T. (2000). Integration of mobile agents with snmp: Why and how. In *Proceedings of the 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS'00)*, páginas 609–622. Honolulu, Havaí, EUA.

[Parker e Pinkas, 1995] Parker, T. e Pinkas, D. (1995). *SESAME Technology Version 4 Overview*.

[Pavlou, 1997] Pavlou, G. (1997). From protocol-based to distributed object-based management architectures. In *Proceedings of the 8th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'97)*, páginas 25–40. Sidney, Austrália.

[Pavlou, 1999] Pavlou, G. (1999). A novel approach for mapping the osi-smi/tmn model for odp/omg corba. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, páginas 67–82. Boston, EUA.

[Pavlou et al., 2004] Pavlou, G., Flegkas, P., Gouveris, S., e Liotta, A. (2004). On management technologies and the potential of web services. *IEEE Communications Magazine*, 42(7):58–66.

[Pavon, 1999] Pavon, J. (1999). Building telecommunications management application with corba. *IEEE Communications Surveys*, 2(2):2–16.

[Postel, 1980] Postel, J. (1980). *User Datagram Protocol - rfc 768*.

[Postel, 1981] Postel, J. (1981). *Transmission Control Protocol - rfc 793*.

[Postel, 1982] Postel, J. (1982). *Simple Mail Transfer Protocol - rfc 821*.

[Postel e Reynolds, 1983] Postel, J. e Reynolds, J. (1983). *TELNET Protocol Specification*.

[Postel e Reynolds, 1985] Postel, J. e Reynolds, J. (1985). *File Transfer Protocol (FTP)*.

[Pras et al., 2004] Pras, A., Drevers, T., Van de Meen, R., e Quartel, D. (2004). Comparing the performance of snmp and web services-based management. *ETransactions on Network and Service Management*.

[Ranc et al., 2000] Ranc, D., Pavlou, G., Griffin, D., e Horra, J. d. l. (2000). Issues and experiences of corba-based management agents. In *Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS'00)*, páginas 447–460. Honolulu, Havaí, EUA.

[Raza et al., 2000] Raza, S., Pagurek, B., e White, T. (2000). Distributed computing for plug-and-play network service configuration. In *Proceedings of the 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS'00)*, páginas 933–946. Honolulu, Havaí, EUA.

[Rescorla, 2000] Rescorla, E. (2000). *HTTP over TLS - rfc 2818*.

[Rivest, 1992] Rivest, R. (1992). *The MD5 Message-Digest Algorithm - rfc 1321*.

[Rose e McCloghrie, 1990] Rose, M. e McCloghrie, K. (1990). *Structure and Identification of Management Information for TCP/IP-based Internets - rfc 1155*.

[Rose, 2001] Rose, M. T. (2001). *The Blocks Extensible Exchange Protocol Core - rfc 3080*.

[Scheifler e Blackman, 2005] Scheifler, R. e Blackman, T. (2005). Página web do projeto davis na comunidade jini. http://davis.jini.org.

[Schönwälder, 2002] Schönwälder, J. (2002). *Simple Network Management Protocol (SNMP) over Transmission Control Protocol (TCP) Transport Mapping - rfc 3430*.

[Schönwälder, 2003] Schönwälder, J. (2003). *Overview of the 2002 IAB Network Management Workshop - rfc 3535*.

[Schönwälder et al., 2003] Schönwälder, J., Pras, A., e Martin-Flatin, J. (2003). On the future of internet management technologies. *IEEE Communications Magazine*, 41(10):90–97.

[Sloten et al., 2004] Sloten, J. v., Pras, A., e Sinderen, M. v. (2004). On the standardisation of web service management operations. In *Proceedings of the 10th EUNICE Summer School (EUNICE'04) and IFIP WG 6.3 Workshop*, páginas 143–150. Tampere, Finlândia.

[Sprenkels e Martin-Flatin, 1999] Sprenkels, R. e Martin-Flatin, J.-P. (1999). Bulk transfers of mib data. *The Simple Times - The Quarterly Newsletter of SNMP Technology, Comment, and Events*, 7(1):1–7.

[Strassner et al., 2004] Strassner, J., Moore, B., Moats, R., e Ellesson, E. (2004). *Policy Core Lightweight Directory Access Protocol (LDAP) Schema - rfc 3703*.

[Strauss e Klie, 2003] Strauss, F. e Klie, T. (2003). Towards xml oriented internet management. In *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management (IM'03)*, páginas 505–518. Colorado Springs, EUA.

[Sun Microsystems, 2001] Sun Microsystems (2001). $Jini^{tm}$ *Technology IP Interconnect Specification - Version 1.0 DraftStandard3*.

[Sun Microsystems, 2002] Sun Microsystems (2002). $Jini^{tm}$ *Technology USB Interconnect Specification - Version 0.1*.

[Sun Microsystems, 2003] Sun Microsystems (2003). $Jini^{tm}$ *Architecture Specification - Version 2.0*.

[The Apache Software Foundation, 2005] The Apache Software Foundation (2005). Página web do apache axis. http://ws.apache.org/axis.

[The Open Group et al., 2000] The Open Group, Object Management Group, e TeleManagement Forum (2000). Inter-domain management: Specification & interaction translation.

[The OpenLDAP Foundation, 2005] The OpenLDAP Foundation (2005). Página web do openldap. http://www.openldap.org/.

[Thompson, 2005] Thompson, K. (2005). Página web do projeto surrogate na comunidade jini. http://surrogate.jini.org/.

[Vambenepe et al., 2005] Vambenepe, W., Graham, S., Askary, S., e Niblett, P. (2005). *Web Services Topics 1.3 (WS-Topics)*. Organization for the Advancement of Structured Information Standards.

[Venners, 2005] Venners, B. (2005). Página web do projeto serviceui na comunidade jini. http://serviceui.jini.org/.

[Vinoski, 1997] Vinoski, S. (1997). Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55.

[Wahl et al., 1997] Wahl, M., Howes, T., e Kille, S. (1997). *Lightweight Directory Access Protocol (v3) - rfc 2251*.

[Waldbusser, 1993] Waldbusser, S. (1993). *Token Ring Extensions to the Remote Network Monitoring MIB - rfc1513*.

[Waldbusser, 1997] Waldbusser, S. (1997). *Remote Network Monitoring Management Information Base Version 2 using SMIv2 - rfc2021*.

[Waldbusser, 2000] Waldbusser, S. (2000). *Remote Network Monitoring Management Information Base - rfc2819*.

[Waldo, 2000] Waldo, J. (2000). The end of protocols. Artigo disponível em http://developer.java.sun.com/developer/technicalArticles/jini/protocols.html.

[Waterman et al., 1999] Waterman, R., Lahaye, B., Romascanu, D., e Waldbusser, S. (1999). *Remote Network Monitoring MIB Extensions for Switched Networks Version 1.0 - rfc2613*.

[Wellens e Auerbach, 1996] Wellens, C. e Auerbach, K. (1996). Towards useful management. *The Simple Times - The Quarterly Newsletter of SNMP Technology, Comment, and Events*, 4(3):1–7.

[Westerinen et al., 2001] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., e Waldbusser, S. (2001). *Terminology for Policy-Based Management - rfc 3198*.

[White, 2000] White, K. D. (2000). *Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations - rfc2925*.

[White et al., 1998] White, T., Bieszczad, A., e Pagurek, B. (1998). Distributed fault location in networks using mobile agents. In *Proceedings of the 3rd. International Workshop on Agents in Telecommunications Applications (IATA'98), AgentWorld'98*, páginas 130–141. Paris, França.

[Wijnen et al., 1998] Wijnen, B., Presuhn, R., e McCloghrie, K. (1998). *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) - rfc 2275*.

[Winer, 1999] Winer, D. (1999). *XML-RPC Specification*. UserLand Software, Inc.

[Yavatkar et al., 2000] Yavatkar, R., Pendarakis, D., e Guerin, R. (2000). *A Framework for Policy-based Admission Control - rfc 2753*.

[Yemini et al., 1991] Yemini, Y., Goldszmidt, G., e Yemini, S. (1991). Network management by delegation. In *Proceedings of the Second IFIP International Symposium on Integrated Network Management*, páginas 95–107. Washington, DC, EUA.

Apêndice A

DTD criada para a definição de Políticas no PNMS expressas em XML

```
<?xml version='1.0' encoding='utf-8'?>
<!ELEMENT policy (condition+, action+, timePeriod)>
<!ATTLIST policy
            name CDATA "default"
            status (enabled|disabled|enabledForDebug) "enabled"
            conditionListType (dnf|cnf) "dnf"
            priority CDATA "0">
<!ELEMENT condition (objectTarget, test)>
<!ATTLIST condition
            name CDATA "default"
            groupNumber CDATA "1"
            negated (true|false) "false">
<!ELEMENT objectTarget (objectTargetID, objectTargetType)>
<!ATTLIST objectTarget wildcarding (true|false) "false">
<!ELEMENT objectTargetID (#PCDATA)>
<!ELEMENT objectTargetType (#PCDATA)>
<!ELEMENT test (existence?, boolean?, threshold?)>
<!ATTLIST test
            frequency CDATA "60"
            delta (true|false) "false">
<!ELEMENT existence EMPTY>
<!ATTLIST existence type (absent|present|changed) "absent">
<!ELEMENT boolean EMPTY>
<!ATTLIST boolean type (unequal|equal|less|lessOrEqual|greater|
greaterOrEqual) "unequal">
<!ELEMENT threshold (thresholdValue)>
<!ATTLIST threshold type (rising|falling|risingOrFalling) "rising">
<!ELEMENT thresholdValue (#PCDATA)>
<!ELEMENT action (set*, notification*)>
<!ATTLIST action
            name CDATA "default"
```

```
            order CDATA "0">
<!ELEMENT set (objectSet+)>
<!ELEMENT objectSet (objectSetID+, objectSetType+)>
<!ELEMENT objectSetID (#PCDATA)>
<!ELEMENT objectSetType (#PCDATA)>
<!ELEMENT notification (objectNotif+)>
<!ATTLIST notification destination CDATA "127.0.0.1">
<!ELEMENT objectNotif (objectNotifID+, objectNotifType+)>
<!ATTLIST objectNotif wildcarding (true|false) "false">
<!ELEMENT objectNotifID (#PCDATA)>
<!ELEMENT objectNotifType (#PCDATA)>
<!ELEMENT timePeriod (initial, final)>
<!ATTLIST timePeriod name CDATA "default">
<!ELEMENT initial (year, month, day, hours, minutes, seconds)>
<!ELEMENT final (year, month, day, hours, minutes, seconds)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day (#PCDATA)>
<!ELEMENT hours (#PCDATA)>
<!ELEMENT minutes (#PCDATA)>
<!ELEMENT seconds (#PCDATA)>
```

Apêndice B

Schema criado para a definição de Políticas no PNMS expressa em termos do PCLS

```
attributetype ( 1.3.6.1.4.1.4203.666.7.1
  NAME 'objectID'
  DESC 'An Object IDentifier (OID).'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.2
  NAME 'objectType'
  DESC 'An OID type.'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.3
  NAME 'wildcarding'
  DESC 'A flag related to usage of wildcarding functionality:
        true - DOES use it; false - does NOT use it.'
  EQUALITY booleanMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.7
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.4
  NAME 'testType'
  DESC 'Type of test to be performed:
        1 - existence; 2 - boolean; 3 - threshold.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.5
```

```
  NAME 'samplingType'
  DESC 'Type of the sampling procedure:
        1 - absolute; 2 - delta.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.6
  NAME 'samplingFreq'
  DESC 'Frequency of the sampling procedure, in seconds.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.7
  NAME 'targetValue'
  DESC 'Numeric value of interest.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.8
  NAME 'existenceType'
  DESC 'Type of existence test applied:
        1 - absent; 2 - present; 3 - changed.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.9
  NAME 'booleanType'
  DESC 'Type of boolean test applied:
        1 - unequal; 2 - equal; 3 - less;
        4 - less or equal; 5 - greater; 6 - greater or equal.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.10
  NAME 'thresholdType'
  DESC 'Type of threshold applied:
1 - rising; 2 - falling; 3 - rising or falling.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

objectclass ( 1.3.6.1.4.1.4203.666.7.11
  NAME 'myConditionAuxClass'
  DESC 'A class representing a trigger in the Event MIB.'
  SUP pcimConditionAuxClass
```

```
  AUXILIARY
  MUST ( objectID $ objectType $ wildcarding $
         testType $ samplingType $ samplingFreq )
  MAY ( targetValue $ existenceType $ booleanType $ thresholdType )
  )

attributetype ( 1.3.6.1.4.1.4203.666.7.12
  NAME 'eventType'
  DESC 'Type of event: 1 - set; 2 - notification;
        3 - set and notification.'
  EQUALITY integerMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.13
  NAME 'objectSetID'
  DESC 'The Object IDentifier (OID) of the MO to be setted.'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.14
  NAME 'objectSetType'
  DESC 'The type of OID of the MO to be setted.'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.15
  NAME 'objectSetValue'
  DESC 'The value of the MO to be setted.'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  SINGLE-VALUE )

attributetype ( 1.3.6.1.4.1.4203.666.7.16
  NAME 'destination'
  DESC 'The IP Address of the host to be notified.'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

attributetype ( 1.3.6.1.4.1.4203.666.7.17
  NAME 'objectNotifID'
  DESC 'The Object IDentifier (OID) of the MO to be
        sent in a notification.'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )

attributetype ( 1.3.6.1.4.1.4203.666.7.18
  NAME 'objectNotifType'
```

```
  DESC 'The type of OID of the MO to be sent in a
        notification.'
  EQUALITY caseIgnoreMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

objectclass ( 1.3.6.1.4.1.4203.666.7.19
  NAME 'myActionAuxClass'
  DESC 'A class representing an event in the Event MIB.'
  SUP pcimActionAuxClass
  AUXILIARY
  MUST ( eventType )
  MAY ( objectSetID $ objectSetType $ objectSetValue $
        destination $ objectNotifID $ objectNotifType )
  )
```

Apêndice C

Arquivo WSDL relativo ao protótipo de Web Service construído

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://143.106.50.214:8080/axis/services/
 WorkStationServer"
 xmlns:apachesoap="http://xml.apache.org/xml-soap"
 xmlns:impl="http://143.106.50.214:8080/axis/services/WorkStationServer"
 xmlns:intf="http://143.106.50.214:8080/axis/services/WorkStationServer"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tns1="urn:AuxClasses"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)-->
 <wsdl:types>
  <schema targetNamespace="urn:AuxClasses" xmlns="http://www.w3.org/2001/
   XMLSchema">
   <import namespace="http://143.106.50.214:8080/axis/services/
    WorkStationServer"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="PrRow">
    <sequence>
     <element name="prMax" type="xsd:int"/>
     <element name="prCount" type="xsd:int"/>
     <element name="prNames" nillable="true" type="xsd:string"/>
     <element name="prErrorFlag" type="xsd:int"/>
     <element name="prErrFix" type="xsd:int"/>
     <element name="prMin" type="xsd:int"/>
     <element name="prErrMessage" nillable="true" type="xsd:string"/>
     <element name="prErrFixCmd" nillable="true" type="xsd:string"/>
     <element name="prIndex" type="xsd:int"/>
    </sequence>
   </complexType>
   <complexType name="PrTable">
```

```
    <sequence>
     <element name="rows" nillable="true" type="impl:ArrayOf_tns1_PrRow"/>
    </sequence>
   </complexType>
   <complexType name="System">
    <sequence>
     <element name="sysName" nillable="true" type="xsd:string"/>
     <element name="sysLocation" nillable="true" type="xsd:string"/>
     <element name="sysContact" nillable="true" type="xsd:string"/>
     <element name="sysDescr" nillable="true" type="xsd:string"/>
     <element name="sysUpTime" type="xsd:long"/>
    </sequence>
   </complexType>
  </schema>
  <schema targetNamespace="http://143.106.50.214:8080/axis/services/
   WorkStationServer"
   xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="urn:AuxClasses"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_tns1_PrRow">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:PrRow[]"/>
     </restriction>
    </complexContent>
   </complexType>
  </schema>
 </wsdl:types>
   <wsdl:message name="getSystemRequest">
   </wsdl:message>
   <wsdl:message name="getSysUpTimeResponse">
      <wsdl:part name="getSysUpTimeReturn" type="xsd:long"/>
   </wsdl:message>
   <wsdl:message name="getSysContactRequest">
   </wsdl:message>
   <wsdl:message name="getSystemResponse">
      <wsdl:part name="getSystemReturn" type="tns1:System"/>
   </wsdl:message>
   <wsdl:message name="getTableResponse">
      <wsdl:part name="getTableReturn" type="tns1:PrTable"/>
   </wsdl:message>
   <wsdl:message name="getSysUpTimeRequest">
   </wsdl:message>
   <wsdl:message name="getSysNameResponse">
      <wsdl:part name="getSysNameReturn" type="soapenc:string"/>
   </wsdl:message>
   <wsdl:message name="getSysLocationResponse">
      <wsdl:part name="getSysLocationReturn" type="soapenc:string"/>
   </wsdl:message>
   <wsdl:message name="getSysDescrRequest">
```

```
</wsdl:message>
<wsdl:message name="getSysNameRequest">
</wsdl:message>
<wsdl:message name="getTableRequest">
</wsdl:message>
<wsdl:message name="getSysDescrResponse">
   <wsdl:part name="getSysDescrReturn" type="soapenc:string"/>
</wsdl:message>
<wsdl:message name="getRowResponse">
   <wsdl:part name="getRowReturn" type="tns1:PrRow"/>
</wsdl:message>
<wsdl:message name="getSysLocationRequest">
</wsdl:message>
<wsdl:message name="getSysContactResponse">
   <wsdl:part name="getSysContactReturn" type="soapenc:string"/>
</wsdl:message>
<wsdl:message name="getRowRequest">
   <wsdl:part name="in0" type="xsd:int"/>
</wsdl:message>
<wsdl:portType name="WorkStationServer">
   <wsdl:operation name="getTable">
      <wsdl:input message="impl:getTableRequest" name="getTableRequest"/>
      <wsdl:output message="impl:getTableResponse" name="getTableResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSystem">
      <wsdl:input message="impl:getSystemRequest" name="getSystemRequest"/>
      <wsdl:output message="impl:getSystemResponse"
       name="getSystemResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSysDescr">
      <wsdl:input message="impl:getSysDescrRequest"
       name="getSysDescrRequest"/>
      <wsdl:output message="impl:getSysDescrResponse"
       name="getSysDescrResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSysUpTime">
      <wsdl:input message="impl:getSysUpTimeRequest"
       name="getSysUpTimeRequest"/>
      <wsdl:output message="impl:getSysUpTimeResponse"
       name="getSysUpTimeResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSysContact">
      <wsdl:input message="impl:getSysContactRequest"
       name="getSysContactRequest"/>
      <wsdl:output message="impl:getSysContactResponse"
       name="getSysContactResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSysName">
      <wsdl:input message="impl:getSysNameRequest"
       name="getSysNameRequest"/>
```

```
      <wsdl:output message="impl:getSysNameResponse"
       name="getSysNameResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getSysLocation">
      <wsdl:input message="impl:getSysLocationRequest"
       name="getSysLocationRequest"/>
      <wsdl:output message="impl:getSysLocationResponse"
       name="getSysLocationResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getRow" parameterOrder="in0">
      <wsdl:input message="impl:getRowRequest" name="getRowRequest"/>
      <wsdl:output message="impl:getRowResponse" name="getRowResponse"/>
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="WorkStationServerSoapBinding"
 type="impl:WorkStationServer">
   <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="getTable">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getTableRequest">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" namespace="http://myExample" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getTableResponse">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" namespace="http://143.106.50.214:8080/axis/
          services/WorkStationServer" use="encoded"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getSystem">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getSystemRequest">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" namespace="http://myExample" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getSystemResponse">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" namespace="http://143.106.50.214:8080/axis/
          services/WorkStationServer" use="encoded"/>
      </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getSysDescr">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getSysDescrRequest">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
          soap/encoding/" namespace="http://myExample" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getSysDescrResponse">
         <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
```

```
        soap/encoding/" namespace="http://143.106.50.214:8080/axis/
        services/WorkStationServer" use="encoded"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getSysUpTime">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSysUpTimeRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://myExample" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="getSysUpTimeResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://143.106.50.214:8080/axis/
       services/WorkStationServer" use="encoded"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getSysContact">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSysContactRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://myExample" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="getSysContactResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://143.106.50.214:8080/axis/
       services/WorkStationServer" use="encoded"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getSysName">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSysNameRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://myExample" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="getSysNameResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://143.106.50.214:8080/axis/
       services/WorkStationServer" use="encoded"/>
   </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getSysLocation">
   <wsdlsoap:operation soapAction=""/>
   <wsdl:input name="getSysLocationRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://myExample" use="encoded"/>
   </wsdl:input>
   <wsdl:output name="getSysLocationResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
       soap/encoding/" namespace="http://143.106.50.214:8080/axis/
       services/WorkStationServer" use="encoded"/>
```

```
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getRow">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getRowRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
             soap/encoding/" namespace="http://myExample" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getRowResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/
             soap/encoding/" namespace="http://143.106.50.214:8080/axis/
             services/WorkStationServer" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="WorkStationServerService">
      <wsdl:port binding="impl:WorkStationServerSoapBinding"
       name="WorkStationServer">
         <wsdlsoap:address location="http://143.106.50.214:8080/axis/
          services/WorkStationServer"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Apêndice D

Publicações relacionadas a Tese

- Helcio W. da Silva e Luís G. P. Meloni: *Gerência Descentralizada de Redes Utilizando Tecnologia JINI e XML* - Conferência Ibero-Americana WWW/Internet (CIAWI) 2003, Algarve-Portugal, 8-9 de Novembro de 2003;
- Helcio W. da Silva e Luís G. P. Meloni: *Using JINI Technology for Building of the Network Management Systems* - IADIS International Conference Applied Computing (AC) 2004, Lisboa-Portugal, 23-26 de Março de 2004;
- Helcio W. da Silva e Luís G. P. Meloni: *Using JINI for Network Management Purposes* - International Workshop on Telecommunications (IWT) 2004, Santa Rita do Sapucaí/MG, Brasil, 23-27 de Agosto de 2004;
- Helcio W. da Silva e Luís G. P. Meloni: *Um Sistema para Gerência de Redes Baseado em Políticas utilizando Tecnologia JINI* - XXII Simpósio Brasileiro de Telecomunicações (SBrT), Campinas-SP, 04 a 08 de Setembro de 2005;
- Helcio W. da Silva e Luís G. P. Meloni: *A Baía de Dispositivos: um NMS Baseado em JINI* - Conferência Ibero-Americana WWW/Internet (CIAWI) 2005, Algarve-Portugal, 18 e 19 de Outubro de 2005;
- Helcio W. da Silva e Luís G. P. Meloni: *Using Surrogates for Network Management* - IV International Information and Telecommunication Technologies Symposium (I2TS), Florianópolis-SC, Brasil, 14-17 de Dezembro de 2005;
- Helcio W. da Silva e Luís G. P. Meloni: *Use of the JINI Technology for Design NMSs: From General Concepts to Prototypes Implementation* - 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), Vancouver, Canadá, 03-07 de Abril de 2006.