HÉLDER HIROFUMI SATO

Um Modelo de Informação de Segurança para Sistemas Distribuídos Baseado em Serviço de Diretórios

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada, Departamento de Ciências Exatas, Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção de título de "Mestre em Ciências" - Área de Concentração: Sistemas Distribuídos

CURITIBA

2000

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
Pró-Reitoria de Pós-Graduação, Pesquisa e Extensão

ATA DA SESSÃO PÚBLICA DE EXAME DE DISSERTAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ.

Exame de dissertação nº 024

Aos 06 dias do mês de outubro de 2000, realizou-se a sessão pública de defesa de dissertação "**Modelo de Informação de Segurança Para Sistemas Distribuídos Baseado de Diretório**", apresentada por Helder Hirofumi Sato, ano de ingresso 1997, para obtenção do título de Mestre em Ciências. A Banca Examinadora foi composta pelos seguintes professores:

MEMBROS DA BANCA	ASSINATURA
Presidente:. **Prof. Dr. Edgard Jamhour (PUCPR)**	
Prof. Dr. Carlos Alberto Maziero (PUCPR)	
Prof. Dr. Elias Procópio Duarte Junior (UFPR)	
Prof. Dr. Raul Fernando Weber (UFRGS)	

De acordo com as normas regimentais a Banca Examinadora deliberou sobre os conceitos a serem atribuídos e que foram os seguintes:

MEMBROS DA BANCA	CONCEITOS
Presidente: **Prof. Dr. Edgard Jamhour (PUCPR)**	APROVADO
Prof. Dr. Carlos Alberto Maziero (PUCPR)	APROVADO
Prof. Dr. Elias Procópio Duarte Junior (UFPR)	APROVADO
Prof. Dr. Raul Fernando Weber (UFRGS)	Aprovado
Conceito Final	

Observações da Banca Examinadora

_____

_____

Profº Júlio Cesar Nievola
Coordenador do Programa de Pós-Graduação em Informática Aplicada–PUCPR

Programa de Pós Graduação em Informática Aplicada
PUC-PR
ÁREA DE INFORMÁTICA

Rua Imaculada Conceição, 1155 – Prado Velho – CEP 80215-901 – Telefone: +55 41 330-1669 / + 55 41 330-1654
Fax: + 55 41 330- 1669 – Curitiba – Paraná – Brasil - E-mail:ppgia@rla01.pucpr.br

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIAÇÕES E SIGLAS

ACE: Access Control Entry
ACL: Access Control List
CIM: Common Information Model
DAP: Directory Access Protocol
DEN: Directory-enabled Networks
DMTF: Distributed Management Task Force
DNS: Domain Name Service
ISP: Internet Solution Provider
LDAP: Lightweight Directory Access Protocol
OSI: Open System Information
QoS: Quality of Service
RFC: Request for Comment
SOR: Sistema Operacional de Rede
SSL: Secure Socket Layer
VPN: Virtual Private Networks

RESUMO

Este trabalho propõe a utilização de um serviço de diretório para dar suporte a aplicações de segurança. O serviço de diretório funciona como um repositório logicamente centralizado, onde as aplicações de segurança podem encontrar informações relativas aos recursos da rede e as regras de segurança que devem implementar.

As informações do diretório são armazenadas na forma de objetos e estruturadas na forma de classes. O conjunto de classes de um diretório é denominado esquema. Este trabalho desenvolve um esquema para dar suporte tanto à serviços básicos de segurança, como autenticação e autorização, como à políticas de segurança mais sofisticadas. A centralização lógica das informações de segurança, decorrente da utilização do serviço de diretório, traz benefícios claros pois os serviços de autorização e autenticação passam a ser realizados de maneira global, isto é, os mesmos mecanismos e informações podem ser compartilhados por diferentes aplicações. Igualmente, pode-se descrever de maneira consistente um conjunto de regras de segurança que deverá ser aplicado a todos os elementos de uma rede e aos recursos compartilhados. Isto significa que o diretório pode representar uma maneira consistente para descrever e armazenar as informações relativas a política de segurança de um ambiente computacional distribuído.

No trabalho desenvolvido nessa monografia, o serviço de diretório funciona como o elemento central de uma arquitetura de segurança. A idéia de utilizar um diretório para auxiliar na administração de sistemas computacionais (não apenas aspectos de segurança) é uma tendência de muitas implementações modernas de sistemas operacionais e sistemas de gerenciamento de redes. Esforços importantes estão sendo feitos com o objetivo de criar um "esquema global" de diretório, onde estariam contemplados os aspectos mais importantes de um

ambiente computacional distribuído. Um desses esforços é o esquema conhecido como DEN (Directory Enable Networks), desenvolvido pelo DMTF (Distributed Management Task Force). Propostas como o DEN são ainda muito recentes, e carecem de uma análise mais aprofundada de sua aplicabilidade. Neste trabalho, o esquema DEN é analisado e novas classes foram incluídas, para dar suporte as informações específicas de segurança. A principal contribuição deste trabalho está no desenvolvimento do esquema de diretório estendido e da definição da dinâmica de utilização dessas informações para implementação dos serviços de segurança.

1. INTRODUÇÃO

Este trabalho endereça o problema da administração da segurança de redes corporativas de grande porte. O ponto principal dessa dissertação é a análise da aplicação de um serviço de diretório como um repositório logicamente centralizado com as informações necessárias para gerenciar a rede. Apesar do serviço de diretório poder ser aplicado a virtualmente qualquer aspecto da gerência de redes [SEM98], esse trabalho concentra-se no subconjunto de informações aplicáveis a gerência da segurança das redes corporativas.

Os ambientes de redes atuais possuem uma grande quantidade de elementos heterogêneos tornando-os complexos de se administrar [ANA97]. Estes elementos podem ser componentes de software como sistemas operacionais, gerenciadores de bancos de dados, ou elementos de hardware como servidores, computadores pessoais, roteadores, switches, hubs, modems.

Além destes elementos serem heterogêneos, eles estão distribuídos fisicamente dentro da estrutura da corporação, o que dificulta ainda mais sua administração, pois na maioria dos casos as informações necessárias para a administração do ambiente ficam situadas localmente em cada elemento de rede. Para que a administração seja facilitada deve existir dentro da infra-estrutura um repositório de informações que permita que se centralize logicamente todas as informações dos elementos de rede e recursos. Desta forma, quando houver a necessidade de se buscar algum dado sobre algum elemento ou recurso isto possa ser executado facilmente.

Este repositório global de informações pode ser implementado utilizando-se um serviço de diretório, que nada mais é do que uma aplicação dentro da infra-estrutura, cuja finalidade básica é prover meios para a criação deste repositório de forma distribuída fisicamente e centralizada logicamente.

Implementando-se o serviço de diretório de forma integrada com as aplicações que dão suporte a gerência de falhas, gerência de configuração, gerência de contabilização, gerência de desempenho e gerência de segurança, a

administração da rede de computadores será facilitada, pois estas aplicações compartilharão a mesma base de informações permitindo que todas trabalhem colaborando umas com as outras.

Este trabalho analisa a aplicação de um subconjunto deste repositório global que dá suporte especificamente às aplicações que implementam serviços de segurança da rede. Deve-se ressaltar, entretanto, que muitos dos conceitos e mecanismos abordados neste trabalho podem ser transportados para as outras áreas de gerência.

O desenvolvimento do trabalho apresentado nesta dissertação está estruturado da seguinte maneira:

No capítulo 2, "Caracterização do Problema", são discutidas as dificuldades encontradas nos ambientes atuais para a implementação e administração da segurança. Neste mesmo capítulo é apresentada uma proposta de uma nova abordagem que utiliza um serviço de diretório para auxiliar os processos referentes a segurança.

No capítulo 3, "Requisitos Necessários", são analisadas as principais características de um serviço de diretório e o papel desempenhado por este componente dentro da abordagem proposta.

No capítulo 4, "Esquema Base para Integração do Serviço de Diretório com a Rede", é analisado um esquema base para o serviço de diretório, ou seja, a estrutura das informações dos principais componentes que devem estar dentro do repositório. O esquema apresentado neste capítulo, serve como modelo base para o esquema de segurança analisado neste trabalho. Este esquema é proposto pelo DMTF (Distributed Management Task Force) e apresentado pela especificação DEN (Directory-enabled Networks)[DEN99].

No capítulo 5, "Análise das Principais Classes do Esquema para Implementações Relativas a Segurança", é analisado um subconjunto do esquema global (modelo composto por todas as classes de objetos do diretório) para o

serviço de diretório que dê suporte aos serviços referentes a segurança de rede como autenticação, autorização e políticas de segurança. Neste capítulo é apresentado um framework que contém um modelo de informação com diagramas de classes que representam a estruturação das informações que devem ser implementadas no serviço de diretório e diagramas de seqüência que demonstram os processos realizados dentro desta estrutura pelas aplicações/serviços de segurança. Também é apresentada uma aplicação do modelo em uma rede corporativa de uma empresa com sites espalhados por várias regiões do Brasil.

No capítulo 6, "Análise da Replicação do Esquema do Diretório", são analisadas as vantagens de se ter uma implementação de um serviço de diretório distribuído pela infra-estrutura da organização onde os dados do repositório são sincronizados através de um processo de replicação de dados.

Para analisar a aplicação de um serviço de diretório integrado com uma aplicação de segurança foi desenvolvido um protótipo que altera permissões de arquivos e diretório buscando os dados armazenados no repositório do serviço de diretório. O código implementado encontra-se no Anexo 3.

2. CARACTERIZAÇÃO DO PROBLEMA

Com o advento de sistemas de informação distribuídos, a necessidade de se implementar técnicas de segurança se tornou vital. Estas técnicas visam estabelecer no ambiente propriedades como confidencialidade, autenticidade, integridade e disponibilidade [GLE97].

Os sistemas de informação atuais, principalmente de grandes organizações, são muito complexos. Administrá-los é uma tarefa desafiadora para os administradores de rede e sistemas, que devem se preocupar com recursos espalhados por diversos locais físicos dentro da organização. Estes recursos (computadores, impressoras, arquivos, etc) são usualmente compartilhados para que mais de uma pessoa possa utilizá-los. Este compartilhamento deve ser controlado, pois nem todos os recursos devem estar disponíveis a todos os usuários da rede. Para que este controle seja realizado, conceitos de segurança como autenticação e autorização [GLE97] são necessários.

A autenticação é a prova da identidade, ou seja, provar quem você diz que é. Para que um usuário se conecte e tenha acesso a rede de computadores este deve se autenticar. Existem várias formas de autenticação que vão desde a posse de alguma coisa que apenas o usuário possui, como um cartão, um disco, até características físicas do próprio usuário como impressão digital, retina, etc.

Após o usuário estar conectado, ou no momento da conexão, existe a necessidade de se estabelecer quais os direitos de acesso deste usuário, ou seja, quais recursos este usuário pode acessar dentro da rede e o que ele pode fazer com cada recurso. Por exemplo, o usuário tem direito de acesso ao arquivo "A" no "Servidor 1" apenas para leitura. O conceito que define este tipo de propriedade é a autorização.

Todas as informações referentes a autenticação e autorização devem estar armazenadas em algum local dentro do sistema.

Em uma configuração como Workgroups (conjunto de computadores que formam um grupo de trabalho) as informações de autenticação e autorização

ficam armazenadas localmente em cada nó (computador conectado na rede que participa do grupo de trabalho), tornando a administração de usuários inadequada para redes de grande porte. Imagine a dificuldade de controle de contas de usuários em um ambiente de grande porte onde, para cada nó que um usuário possa utilizar, deva-se criar uma conta e definir suas autorizações.

A figura 2.1 mostra esta configuração. Como em cada nó participante do grupo de trabalho as informações de autenticação e autorização são armazenadas localmente, um usuário que deseja utilizar todos os nós deve estar cadastrado nos nós A, B e C.

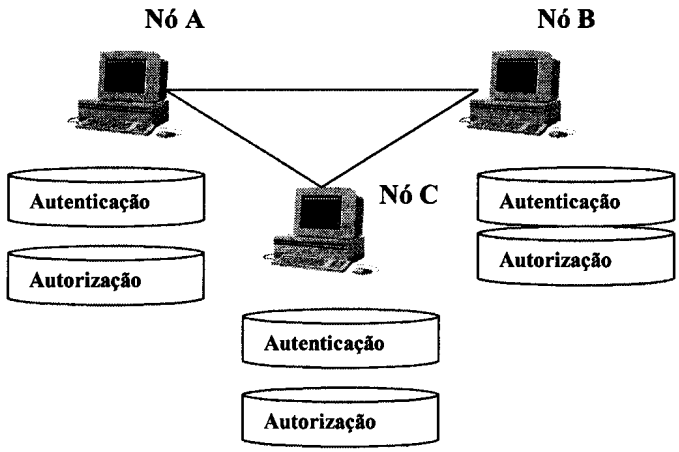

Figura 2.1 - Informações de Segurança em uma Configuração Workgroups.

Já em um ambiente de domínios (conjunto de computadores em um mesmo grupo administrativo) as informações de autenticação ficam armazenadas em um único ponto (podendo ser replicadas), mas as informações de autorização continuam locais em cada nó. Com isto, facilita-se a administração de contas de usuários mas continua existindo a dificuldade de se controlar a autorização dos recursos compartilhados na rede.

A figura 2.2 mostra uma configuração de domínios. Note que o nó A tem a responsabilidade de armazenar as informações de autenticação de todos os membros do grupo administrativo. Desta forma, cada vez que um usuário se

conecta no nó A, B ou C, suas informações para autenticação serão verificadas no nó A.

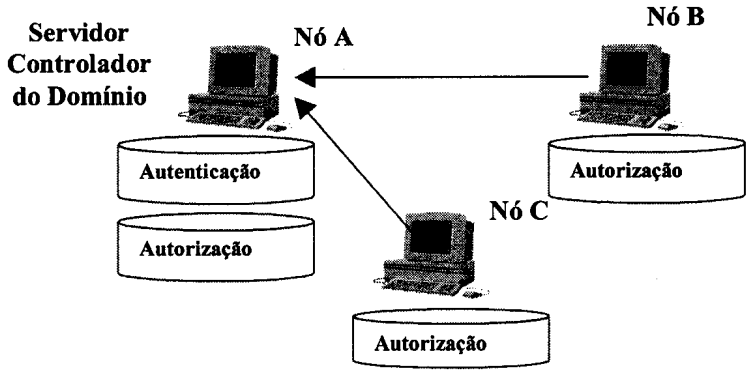

Figura 2.2 - Informações de Segurança em uma Configuração de Domínios.

Um diretório ou serviço de diretório, pode ser definido genericamente como sendo o local onde as informações que dão suporte às aplicações da rede ficam armazenadas. Por exemplo, pode existir um diretório para mapeamento de contas de usuários, um diretório para dar suporte a um serviço de resolução de nomes como DNS (Domain Name System) [TAN96], etc.

Um problema encontrado nas implementações atuais, é que muitos serviços de diretório são implementados independentemente para cada aplicação, prejudicando a administração do sistema. Este problema é mostrado na figura 2.3.

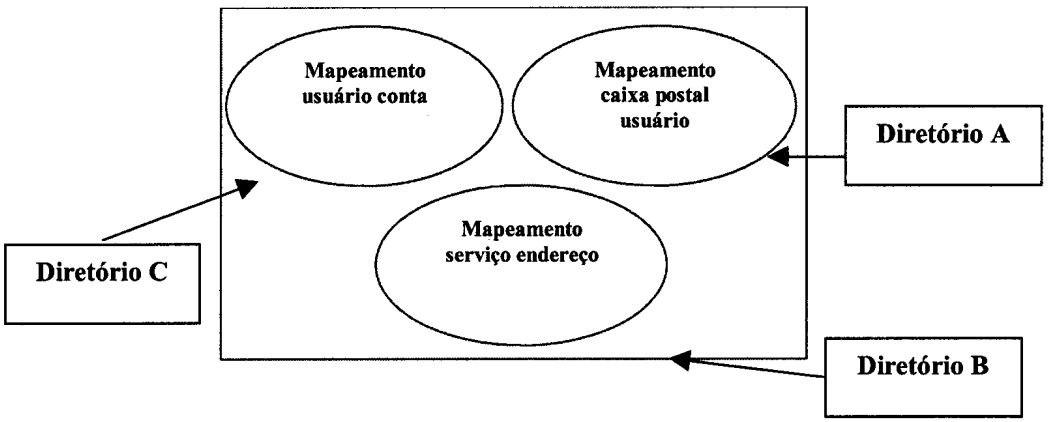

Figura 2.3 - Implementação de um Serviço de Diretório Independente para cada Aplicação

A figura 2.3 mostra que as informações de contas de usuários, de caixa postal de usuários e de endereçamento de rede são armazenadas em diretórios distintos. Neste tipo de implementação fica evidente o problema de duplicação de informações que acontece nos diversos diretórios mantidos pelas diversas aplicações, tornando a administração cada vez mais complexa, pois pode ficar difícil manter a integridade das informações em uma estrutura de grande porte.

Este trabalho analisa um modelo genérico de segurança para sistemas de informação distribuídos que utiliza um serviço de diretório global integrado com o serviço de segurança. Um serviço de diretório global é um repositório de informações centralizado logicamente utilizado para armazenar informações sobre os recursos do sistema.

Diferente das implementações anteriores, onde o diretório possui informações apenas sobre contas de usuários (modelo de domínios, figura 2.2), esta abordagem prevê que no serviço de diretório estarão armazenadas as políticas de segurança e as informações sobre as contas de usuários e recursos.

A figura 2.4 mostra o modelo onde todas as informações referentes a segurança ficam armazenadas em um repositório comum. Note que este repositório pode ser replicado ou distribuído por motivos de desempenho e alta disponibilidade.

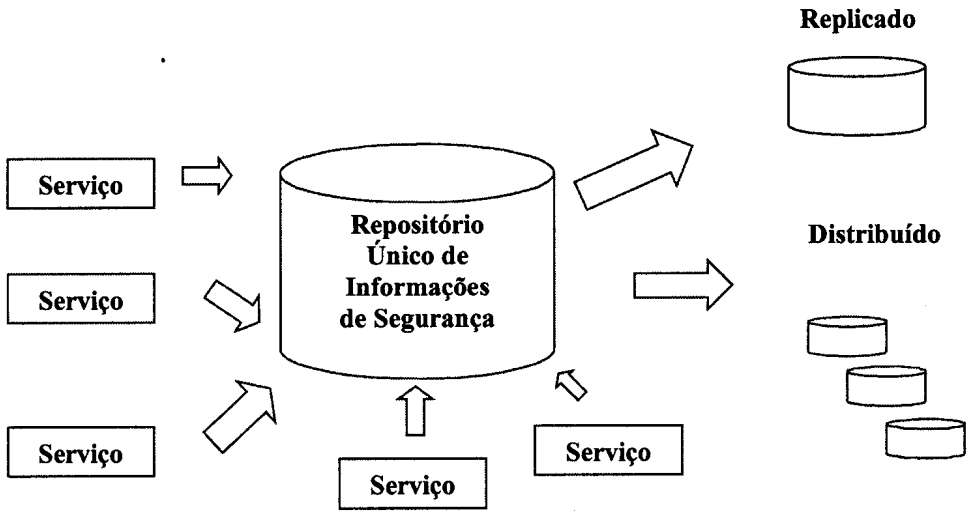

Figura 2.4 - Implementação de um Serviço de Diretório Global

Neste modelo, as dificuldades de administração decorrentes da duplicação de informações desaparecem.

Para acesso às informações armazenadas no serviço de diretórios deve-se utilizar um protocolo padrão, para que assim se alcance a interoperabilidade entre sistemas de diferentes plataformas. O protocolo que atualmente possui estas características é o Lightweight Directory Access Protocol (LDAP) [RFC1777] que agora está em sua versão 3 [RFC2251]. Este protocolo aberto, definido por RFCs, é compatível com a Internet, isto é, permite que aplicações efetuem autenticação e busca de informações através de uma rede TCP/IP. Com isto, consegue-se um nível de segurança que contemple todos os recursos compartilhados disponibilizados no sistema.

Como principais vantagens a serem analisadas podem-se destacar: o controle de informações de segurança de todos os recursos da rede estão armazenadas em um único espaço lógico, fornecendo um ponto central para a administração da segurança; o controle de segurança é feito através de atributos (dentro de um serviço de diretório os recursos são tratados como objetos que possuem atributos e estes podem receber níveis de autorização diferenciados); a procura de recursos é facilitada, pois pode-se ter uma visão global de todos os recursos com suas autorizações dentro do sistema.

Desta forma, implementações de redes baseadas em políticas [SEM98] poderão ser efetuadas. Neste tipo de implementação todas as configurações e políticas relacionadas a configuração, acesso e controle dos recursos da rede ficam armazenadas no repositório do servidor de serviço de diretório. Com estas informações centralizadas, pode-se implementar processos de segurança que manterão os recursos automaticamente no que se refere a administração. Por exemplo, quando um novo recurso for adicionado a rede (servidor, roteador, switch, etc) um serviço ou uma aplicação de gerenciamento de segurança irá buscar as configurações de segurança que se aplicam a este recurso em um serviço de diretório global e, automaticamente, realizará as configurações necessárias sem a intervenção de um administrador.

Um outro exemplo, é a definição de uma política de que a partir das 10 horas da noite não se deve ter acesso a uma determinada aplicação. Com as informações referentes a esta política armazenadas no servidor de serviço de diretório, quando algum usuário tentar acessar a aplicação, um processo previamente implementado poderá verificar a política no diretório global, cruzar estas informações com as informações de horário obtidas a partir de um servidor de tempo (*Time Server*) e não permitir que o acesso seja realizado.

Para que implementações deste tipo possam ser realizadas existe a necessidade de formulação de um modelo genérico que demonstre como deve ser projetada a segurança baseada em um serviço de diretório global. Este modelo terá como principais preocupações a independência de administração, a autenticação global e um modelo de como as informações de segurança poderão ser armazenadas e mantidas. A definição deste modelo deve ser realizada partindo-se do princípio de que toda a estrutura organizacional dos sistemas de informação deve ser mapeada utilizando-se o conceito de diretório. Após este mapeamento, deve ser analisado como os conceitos de autenticação, autorização e políticas de segurança se enquadram nesta estrutura para que sua aplicação atenda a todas as necessidades da organização.

3. REQUISITOS NECESSÁRIOS

Serviços de Diretório

Para formular o modelo de informação para aplicação da segurança baseada em um serviço de diretório é necessário analisar as principais características deste tipo de serviço.

Os serviços de diretórios são ferramentas que gerenciam informações de usuários e recursos da rede. Um serviço de diretório é baseado em um esquema e um protocolo de acesso. Um esquema é uma base de dados que contém os objetos de uma rede (usuários, impressoras, servidores, etc) e seus atributos (nome do usuário, privilégios de acesso, endereço IP de impressoras, etc). A base de dados de um diretório pode ser baseada em um esquema proprietário ou em um esquema de um padrão aberto. O mesmo acontece com o protocolo de acesso ao diretório. Características como criação e gerência de informações de usuários/recursos, capacidade de se executar consultas na base de dados, autenticação e autorização são requisitos mínimos para um serviço de diretório.

Geralmente, os serviços de diretório vêm incorporados ao sistema operacional de rede (SOR). Contudo, esta implementação pode ser menos benéfica caso a rede possua vários SORs que armazenem as informações do diretório utilizando um esquema proprietário, ou seja, não global. A meta a ser alcançada por um serviço de diretório é ser um metadiretório, ou seja, gerenciar múltiplos SORs heterogêneos, múltiplas aplicações e ambientes como a Internet ou Intranets.

Os padrões abertos estão sendo as diretrizes chaves na definição dos serviços de diretórios. Um serviço de diretório global deve ser baseado em um método conhecido e uniforme de acesso aos seus dados. Embora, por baixo da base de dados do diretório, as informações sejam armazenadas em um formato proprietário, desenvolvedores devem ser capazes de acessar o diretório utilizando um método padrão.

O padrão X.500 [ITUX500] define um esquema e um protocolo de acesso ao diretório (DAP - Directory Access Protocol) e é dividido em Directory Model, Information Model e Security Model. Este padrão tem a intenção de fornecer um esquema comum para armazenar informações de um diretório global. Embora o X.500 seja um padrão completo, ele não foi amplamente adotado. O esquema de diretório é freqüentemente adotado mas o protocolo de acesso DAP, mesmo sendo rico em características e executado sobre a pilha de protocolos de rede OSI, fornece um acesso ao diretório lento e está sendo considerado inviável para as implementações atuais.

Por isso, o LDAP [HOWE97] (Lightweight Directory Access Protocol) vem sendo amplamente utilizado. O LDAP simplifica algumas das funções mais complexas desempenhadas pelo DAP e fornece um acesso mais rápido, pois foi projetado para ser executado diretamente sobre a pilha TCP/IP.

A segurança é um requisito básico para serviços de diretório, e um grande número de padrões de segurança existem. Contudo, muitos fornecedores utilizam uma combinação de métodos padrões e proprietários. Um dos mecanismos de segurança mais utilizados é o SSL (Secure Socket Layer) [NET96] e é projetado para fornecer privacidade entre duas aplicações que se comunicam (um cliente e um servidor). O SSL requer um protocolo de transporte como o TCP para transmissão de dados. Para autenticação em um serviço de diretório pode ser utilizado servidores de autenticação com o Kerberos [RFC1510].

A funcionalidade e as características de um serviço de diretório estão diretamente relacionadas a sua arquitetura. Os serviços de diretórios que seguem o padrão X.500 são construídos segundo uma arquitetura hierárquica, o que torna sua implementação flexível e escalável. Esta arquitetura pode suportar deste uma pequena organização até requisitos de Internet de um ISP (Internet Service Provider). A arquitetura hierárquica é flexível o bastante para suportar os requisitos multi-árvore de um metadiretório.

A tabela 3.1 descreve as principais arquiteturas de diretórios:

Arquitetura	Descrição
Server-Based (Baseada em Servidor):	Cada servidor é administrado separadamente de todos os outros servidores. Para se utilizar um recurso de outro servidor, deve-se conhecer qual servidor possui o recurso e como se conectar naquele servidor. Esta arquitetura é muito limitada pois não possui boa escalabilidade e nem fornece flexibilidade.
Domain (Domínio):	Esta arquitetura agrupa logicamente cada servidor e todos os usuários/recursos de rede associados em um domínio. Arquiteturas baseadas em domínios são mais difíceis de se gerenciar do que as hierárquicas.
Hierárquica:	Esta arquitetura utiliza uma estrutura de árvore para representar o diretório. Ramos da árvore representam divisões, departamentos ou locais. Folhas representam objetos de redes individuais como usuários, servidores e impressoras. Privilégios de acesso e outras informações podem ser atribuídas para ramos inteiros facilitando a administração. Se implementada de forma correta, esta arquitetura é muito flexível e escalável podendo suportar desde redes pequenas até redes globais com a Internet.
Application-Based (Baseada na Aplicação):	Esta arquitetura geralmente é proprietária e não é projetada para ser flexível e escalável. Também conhecidas como white pages, mantém informações sobre aplicações e usuários específicos.
Metadiretórios:	Um metadiretório permite um único ponto de administração para múltiplos serviços de diretório baseados em SORs ou aplicações. Um metadiretório fornece interoperabilidade entre sistemas operacionais heterogêneos em uma rede.

Tabela 3.1 - Principais Arquiteturas de Serviços de Diretórios

Dentre as principais metas dos serviços de diretório pode-se citar:

- Fornecer aos administradores de redes a capacidade de gerenciar usuários e recursos através de toda a rede (criar/alterar/eliminar objetos, organizá-los em grupos, atribuir privilégios, combinar árvores, combinar ramos, mover ramos ou folhas, etc).
- Implementar login único, que permite aos usuários serem autenticados em qualquer ponto da rede.
- Permitir a incorporação de características de metadiretórios para fornecer interoperabilidade entre diferentes sistemas operacionais.
- Possuir sistemas de replicação eficientes permitindo que a base de dados seja distribuída globalmente.
- Possuir uma base de dados distribuída para fornecer um crescimento ilimitado.
- Permitir que se estenda o esquema padrão, para incluir novos objetos/atributos na base de dados.
- Possuir um interface gráfica de fácil manuseio.

Papel do Serviço de Diretório

O serviço de diretório é um repositório de informações logicamente centralizadas e fisicamente distribuídas. Os dados contidos no diretório, diferente de bancos de dados tradicionais, sofrem pouca alteração e são utilizados para gerenciar todo o ambiente da rede.

O repositório de informações do serviço de diretório contém informações de todos os recursos disponibilizados no ambiente distribuído, facilitando assim a pesquisa de qualquer objeto da rede que pode ser localizado e identificado.

A tecnologia de serviços de diretórios deve ser projetada para atender as complexas aplicações existentes, onde o serviço de diretório assuma o papel de um repositório de informações inteligente e distribuído.

Desta forma, ele será fundamental para a implantação de uma infra-estrutura onde os recursos da rede (aplicações, serviços, etc) de acordo com situações momentâneas poderão se configurar automaticamente ou obedecer a determinadas políticas para que os processos sejam executados de forma apropriada. Este dinamismo só poderá ser alcançado através da utilização de um servidor de diretório global, onde as informações de todos os objetos envolvidos no processo estejam disponíveis em um repositório logicamente centralizado. Com isso, a definição de políticas por parte do administrador para controlar os dispositivos e serviços da rede poderá ser realizada.

Em termos gerais, as políticas devem estabelecer quais recursos um dado consumidor pode utilizar no contexto de uma dada aplicação ou serviço.

Para a implementação de um serviço de diretório que atenda as necessidades dos elementos da rede e seus usuários, deve-se definir como as informações serão agrupadas e correlacionas, ou seja, deve ser definido um modelo de informação.

Para maiores informações sobre as implementações de servidores de serviço de diretório comerciais veja [ADS97], [NDS98] e [NET97].

Modelo de Informação

Um modelo de informação é uma abstração do conhecimento para que este possa ser entendido antes de ser implementado.

O modelo de informação estrutura o conhecimento sobre os objetos da rede (usuários, aplicações, serviços, dispositivos, etc) e como eles interagem em múltiplos domínios para permitir que diferentes pessoas possam utilizá-los. Esta estruturação é feita utilizando-se um esquema orientado a objetos.

Este esquema consiste em um conjunto de classes abstratas de onde serão derivadas subclasses concretas que representarão os objetos reais da estrutura computacional.

Diferente de usuários e servidores, que podem ser considerados objetos estáticos, dispositivos de rede e os serviços executados na rede são objetos

complexos que residem em um ambiente que constantemente se altera, portanto sua descrição é complexa. Apenas a hierarquia de classes é insuficiente para descrever um elemento ou um serviço de rede, pois a hierarquia de classes não modela a interação existente entre os elementos de rede e seu comportamento individual ou em grupo.

No mínimo dois aspectos devem ser modelados em sistemas complexos:

Modelo de Objetos

Este modelo deve descrever a estrutura estática (dados, atributos, operações e relacionamentos com outros objetos) do sistema.

Modelo Dinâmico

Este modelo deve descrever relacionamentos temporais (comportamento) entre diferentes componentes do sistema e como cada componente é controlado em função do tempo, ou seja, um modelo dinâmico examina as modificações ocorridas nos objetos e seus relacionamentos em relação ao tempo.

O modelo dinâmico especifica quando acontece e o modelo de objetos o que acontece a quem.

Neste trabalho é proposto um esquema base extensível do qual poderão ser derivados objetos particulares que tem papel fundamental na segurança do sistema. Um esquema base extensível é um conjunto de classes padrões que permitem que outras classes herdem suas carcterísticas e implementem apenas atributos particulares (Por exemplo, as classes automóvel e aeronave herdam as características da classe veículo e implementam atributos particulares de cada uma). Para isto, é analisado um modelo de objetos que é composto de definições do esquema X.500, definições do esquema CIM (Common Information Model) [CIM99], definições do esquema DEN (Directory-enabled Networks) [DEN99] e definições próprias deste trabalho.

O CIM é um modelo de dados de um esquema neutro (independente de fornecedor) que descreve todas as informações de gerenciamento em um ambiente de redes de uma corporação. O CIM é composto por uma especificação e um esquema. A especificação define os detalhes de integração com outros modelos de gerenciamento (Ex.: MIBs do SNMP, etc) enquanto que o esquema é o próprio modelo.

A especificação DEN foi projetada para fornecer a base para redes mais inteligentes pois mapeia usuários e serviços de rede e mapeia critérios do negócio em relação ao serviços da rede.

A especificação CIM e a especificação DEN são de responsabilidade do DMTF (Distributed Management Task Force). O DMTF é uma organização que conduz o desenvolvimento, a adoção e a unificação de padrões de gerenciamento. Esta organização trabalha com fornecedores chaves de tecnologia e grupos de padronização afiliados. Maiores informações sobre o CIM e o DEN podem ser buscada no site da Internet do DMTF (www.dmtf.org).

Deve-se notar que o esquema analisado é baseado em padrões abertos, pois este esquema deve dar suporte a qualquer componente da arquitetura sem importar o fornecedor, para que assim possa ser implementado em qualquer ambiente onde seja necessário.

Um modelo dinâmico será apresentada para as situações que envolvem a autenticação, autorização, políticas relativas a segurança permitindo assim a aplicação de uma arquitetura de segurança distribuída baseada no serviço de diretório.

4. ESQUEMA BASE PARA INTEGRAÇÃO DO SERVIÇO DE DIRETÓRIO COM A REDE

Um esquema define quais objetos podem ser criados em um diretório e quais atributos são utilizados para descrever estes objetos. A definição de um objeto dentro do diretório (na verdade a definição de uma classe de objetos, pois objetos são apenas instâncias destas classes) deve conter uma lista de atributos obrigatórios da classe para que esta possa ser armazenada no diretório, uma lista de atributos adicionais e uma lista de classes da qual a classe pode ser derivada (normalmente o esquema é baseado em herança simples).

O esquema adotado como base na construção do esquema proposto por este trabalho é o definido pela especificação DEN [DEN99] que é responsabilidade do DMTF. É importante frisar que o esquema do DEN serve apenas como uma base, pois através deste trabalho serão definidas novas classes, novos atributos e relacionamentos.

O esquema proposto aqui poderá ser implementado em qualquer serviço de diretório que suporte LDAP (Light Directory Access Protocol) e extensibilidade (permita a criação de novas classes dentro do esquema do repositório).

Este esquema consiste em um conjunto de classes abstratas e algumas classes concretas. Classes abstratas não possuem instâncias diretas de objetos e são utilizadas de forma que as suas subclasses herdem seus atributos e métodos [RUM91].

O esquema DEN fornece uma forma de organizar os elementos de rede e as políticas que os gerenciam. Este esquema, desenvolvido pelo DMTF, sofre influências do X.500 e do CIM [CIM97] além de definir suas próprias classes, realiza alterações nas classes herdadas dos outros esquemas. A especificação DEN se preocupa com a integração e a interoperabilidade entre o serviço de diretório e as aplicações da rede, fornecendo um *framework* para

que estas aplicações se comuniquem com os dispositivos de rede (switch, roteador, etc) através de um protocolo comum. Esta comunicação permite que as operações de rede sejam otimizadas.

Quando se fala em DEN dois termos se destacam: profile e políticas.

Os profiles são utilizados para abstrair o conceito de usuários, serviços e aplicações. Um profile é um modelo que contém as características e o comportamento de um objeto ou um conjunto de objetos e podem ser aplicados a um usuário ou a um grupo de usuários fornecendo assim um alto nível de abstração. Em resumo, os profiles dizem ao sistema o que deve ser feito, mas não os passos necessários para fazê-lo.

As políticas definem o comportamento desejado entre dois ou mais objetos. É importante notar que política não quer dizer auditoria nem execução (coação). Em uma rede, políticas se aplicam a serviços como: configuração, roteamento, chaveamento (switching), controle de acesso e uso dos serviços como encriptação e QoS (Quality of Service) [FER98]. Um modelo de políticas centralizadas pode ser a chave para um gerenciamento eficiente da rede, pois a implementação consistente de políticas é uma tarefa árdua devido a complexidade de se gerenciar uma grande quantidade de características de diversos tipos de elementos de rede.

A especificação DEN define uma forma padrão de armazenar políticas e profiles bem como o modelo de informação que define as interações entre objetos da rede. O serviço de diretório também representa os consumidores, isto é objetos que consomem os recursos da rede como usuários e aplicações, permitindo que aplicações compatíveis com DEN e os próprios elementos da rede descubram e executem políticas no ponto onde os recursos são consumidos. Isto permite que com a aplicação do esquema da especificação DEN seja possível realizar um controle de políticas a nível de usuário ou grupo de usuários.

A especificação define um conjunto de modelos de dados para representar os elementos da rede e implementa estes modelos como extensões do esquema de diretório. Devido ao DEN suportar informações da

rede no diretório, pode-se associar usuários específicos com propriedades da rede, permitindo assim a configuração dinâmica da rede para suportar cada necessidade do usuário. Por exemplo, suponha que um determinado usuário possa utilizar a rede para vídeo conferência. Não importa onde este usuário se conecte na rede, ele necessita de caminhos seguros para a conferência. Com o esquema DEN implementado em um serviço de diretório, uma aplicação de gerenciamento pode acessar as informações necessárias para definir quais serviços e métodos devem ser utilizados para disponibilizar o caminho seguro para que o usuário realize sua vídeo conferência.

A figura 4.1 abaixo demonstra as classes de objetos básicas no esquema DEN:

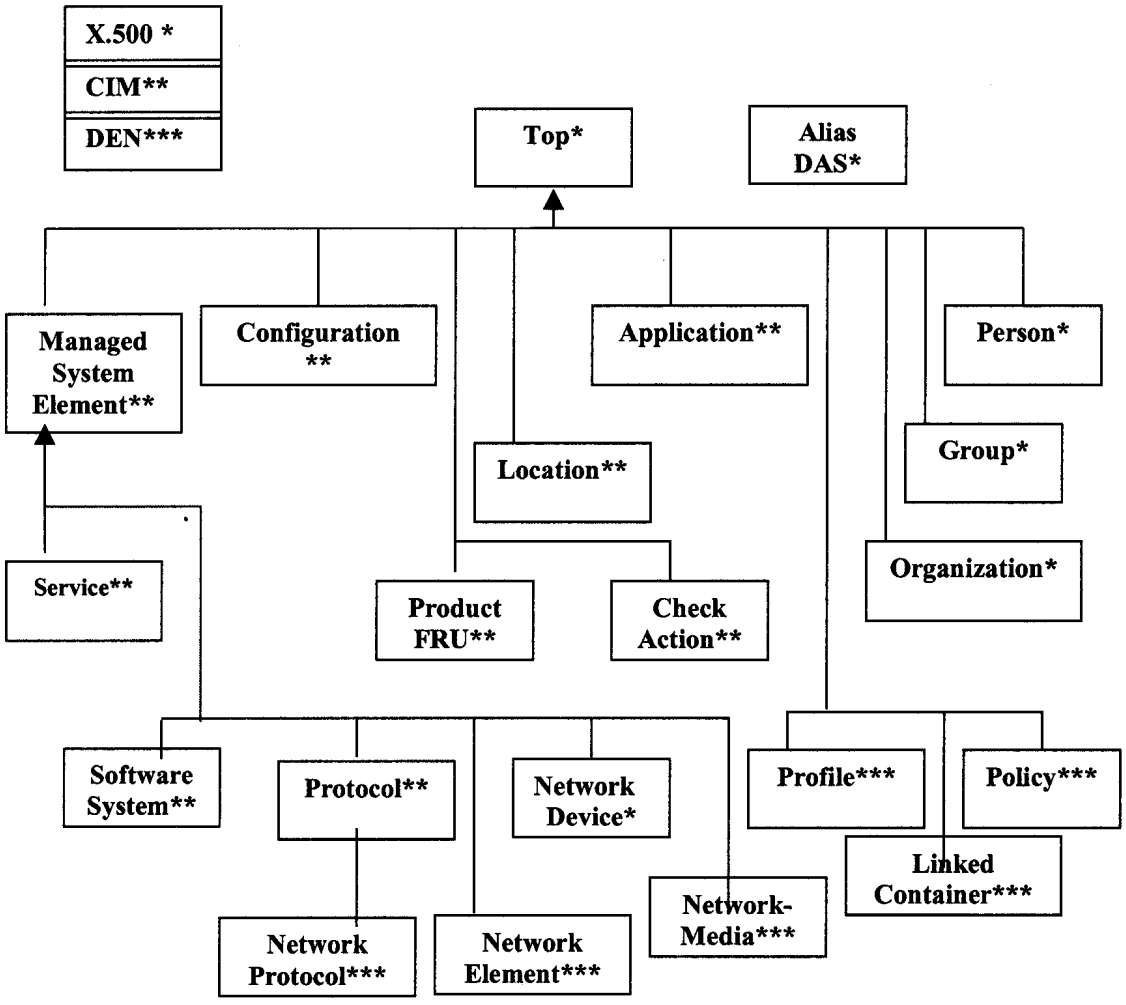

Figura 4.1 - Estrutura funcional das classes bases do DEN

Note na figura 4.1 acima que o esquema DEN é composto por classes definidas pelo X.500 (marcadas com *), classes definidas pelo CIM (marcadas com **) e classes definidas pelo próprio DEN (marcadas com ***). No início da hierarquia estão as classes X.500 e CIM que definem as informações dos elementos físicos e lógicos a serem gerenciados. As classes DEN completam o modelo CIM permitindo que se modele elementos de rede, serviços de rede e políticas. Uma breve descrição das classes da figura 4.1 é apresentada mais adiante neste capítulo.

Com a aplicação do esquema DEN, o diretório se torna um repositório para as informações de todos os elementos de rede sem importar qual o fornecedor e o tipo do elemento. Este repositório comum permite que se faça análises mais sofisticadas para retirar o máximo proveito do ambiente. Além disso, configurações e políticas consistentes poderão ser aplicadas de forma automática por toda a empresa, caso se implemente uma aplicação de gerenciamento de políticas que busque informações no diretório. Por exemplo, pode-se criar uma política no diretório para bloquear pacotes UDP e então aplicar esta política em todos os roteadores de um determinado domínio administrativo. A vantagem de utilizar um método como este, é que a política será aplicada sem importar o fornecedor de hardware ou a versão do software administrativo do dispositivo. A aplicação de gerenciamento compatível com DEN acessará o diretório para buscar as regras de configuração no objeto de políticas e os métodos de configurações específicas do dispositivo no objeto do dispositivo de rede. Desta forma a aplicação de gerenciamento poderá alterar as configurações do dispositivo de acordo com as regras que a política define, utilizando os métodos de configuração apropriados para o dispositivo específico. Para aplicar estas configurações no dispositivo a aplicação de gerenciamento poderá utilizar protocolos de aplicação como telnet, SNMP ou até mesmo o LDAP, dependendo do suporte à administração que o dispositivo fornecer.

O resultado obtido é a aplicação de políticas de forma consistente e automatizada até mesmo em ambientes heterogêneos.

O que já está sendo feito pelos fornecedores de hardware é o desenvolvimento de dispositivos que utilizam as vantagens da especificação DEN. Estes dispositivos poderão receber informações de configuração direto dos objetos de políticas armazenados no diretório.

Este nível de integração permitirá a execução de políticas de QoS (Quality of Service) dinamicamente baseadas no logon do usuário (informações de identificação do usuário), saturação da largura de banda ou outros eventos.

Outras implementações que poderão ser realizadas são fornecimento de largura de banda ponto-a-ponto e segurança baseada em aplicações ou políticas de usuários e serviços.

Abaixo estão listadas as principais classes da especificação DEN. Para maiores informações consulte as especificações X.500, CIM e DEN ou o anexo 1 deste trabalho.

Influências do X.500

O X.500 fornece os seguintes conceitos para o DEN:

Top: Classes base que é o ponto de início para todas as outras classes modeladas.

Person: Refere-se as classes *Person*, *OrganizationalPerson* e ResidentialPerson. Estas classes são utilizadas para definir conceitos genéricos de pessoas associadas, como empregados de um negócio. O DEN utiliza estas classes para dois propósitos: para representar os clientes de serviços da rede e para representar o proprietário/administrador de um dispositivo ou serviço.

Group: O X.500 define duas classes - Grupo de Nomes (Group of Names) e Grupo de nomes Únicos (Group of Unique Names) - que descrevem um conjunto desordenado de nomes que representam objetos individuais ou outros grupos de nomes.

Organization: O X.500 define três classes - *Organization*, *OrganizationUnit* e *OrganizationRole* que representam respectivamente:

negócios, subdivisões de negócios e posições (papéis) de uma organização. O DEN estende estes conceitos para que dispositivos e serviços pertençam a entidades de negócio.

Application: Tanto o X.500 quanto o CIM definem o conceito de *Application* (aplicação). O X.500 define as classes *ApplicationProcess* e *ApplicationEntity*. O DEN adiciona algumas informações para que estas classes possam ser associadas a elementos de redes e serviços.

Influências do CIM

O CIM fornece os seguintes conceitos para o DEN:

Product, etc: O CIM define esta coleção de classes para representação de um produto e suas partes substituíveis. Estas classes são colocadas como complemento para que aplicações (como inventário, capacity planning, etc) baseadas em DEN possam utilizá-las.

ManagedSystemElement: O CIM utiliza esta classe para representar qualquer sistema ou componente do sistema que deva ser gerenciado. Esta classe é super classe de *PhysicalElement* e *LogicalElement* que são classes bases para definição de características físicas e lógicas dos componentes do sistema.

Configuration: O CIM define uma classe *Configuration* e uma classe *Setting*. A classe *Configuration* representa um grupo de "settings" que coletivamente representam um certo comportamento ou estado funcional de um objeto *ManagedSystemElement*. O DEN estende este conceito para modelar a configuração de elementos de rede e serviços.

Service: O CIM define dois conceitos importantes, *Service* e *ServiceAccessPoint*, ambos críticos para definição de serviços de rede. Um serviço é definido como um elemento lógico que contém informação necessária para representar e administrar a funcionalidade fornecida por um dispositivo e/ou característica de software. Um Service é um objeto de propósito geral para configurar e administrar a implementação de uma funcionalidade. Um

objeto *ServiceAccessPoint* é definido como a habilidade de se utilizar e invocar um serviço. Estes dois conceitos são derivados da classe *LogicalElement.*

Software: O CIM define uma noção geral de software utilizando as classes *SoftwareFeature* e *SoftwareElement*, ambas derivadas da classe *LogicalElement.* A classe *SoftwareFeature* é utilizada para descrever uma funcionalidade ou capacidade de um produto ou aplicação. A classe *SoftwareElement* representa a porção administrativa de um objeto *SoftwareFeature*. O DEN refina este conceito, em conjunto com *System*, *Check* e *Action* para alcançar requisitos específicos de elementos de rede e serviços.

System: O CIM define uma rica hierarquia para sistema. A classe *System* é derivada de *LogicalElement* e representa uma agregação de um conjunto de objetos que operam funcionalmente como um todo. A Classe *ComputerSystem* é derivada de *System* e representa uma coleção de componentes que possuem capacidade computacionais servindo como ponto de agregação para componentes como sistemas de arquivos e sistemas operacionais. O DEN utiliza os conceitos de *System*, *ComputerSystem*, *OperatingSystem*, *FileSystem* e outros para representar o conceito de elementos de rede lógicos.

Location: O CIM define a classe *Location* que especifica o endereço e a localização de um elemento físico.

Check e *Action*: Classes "top" utilizadas por objetos *SoftwareElement.* Um *Check* é uma condição ou característica que se espera ser verdadeira para um dado sistema computacional. Uma ação é uma operação que pode alterar o estado de um elemento de software

Application: O CIM também define a noção de aplicação, adicionando o conceito de que uma aplicação é utilizada para suportar uma determinada função do negócio e como tal ela pode ser gerenciada como uma unidade individual.

Novas Classes DEN

DEN estende/melhora os seguintes conceitos:

NetworkService: Ponto de partida para a hierarquia de serviços de rede. Esta classe é derivada da classe CIM *Service*. Ela define várias associações importantes entre várias outras classes.

NetworkProtocol: Ponto de partida para todas as outras classes de protocolo de rede. A classe básica *Protocol* é utilizada como âncora para a árvore de protocolos. Uma classe *ProtocolEndPoint* é utilizada para descrever o ponto de comunicação do qual dados podem ser enviados e recebidos.

Melhorias para *PhysicalPackage* e *Card*: O X.500 define o conceito genérico de *Device* (dispositivo). Contudo, o X.500 não fornece uma definição concreta para que serve um *Device* nem define nada sobre elementos de rede. O CIM possui uma rica definição de um componente físico mas não define o conceito de dispositivo de rede. O DEN estende e melhora as classes do CIM *PhysicalPackage* e Card para incluir as funcionalidades necessárias para os dispositivos de rede. Novos relacionamento também são necessários para associar dispositivos de rede com outras classes.

NetworkElement: O X.500 define o conceito genérico de um dispositivo, mas não define sua funcionalidade lógica. O CIM possui esta definição mas não para dispositivos de rede. O DEN estende e melhora a definição CIM (mantendo compatibilidade com X.500) de elementos lógicos para incluir elementos de rede. Os aspectos lógicos de um elemento de rede são representados na classe *NetworkElement*. Novos relacionamentos também são necessários para associação com outras classes.

A especificação DEN define novos conceitos:

Profile: Classe que possui informações referentes a características e necessidades de usuários, aplicações e demais elementos de rede. Esta

classe deve ser especializada para caracterizar o que objetos particulares podem realizar em um contexto particular sujeitos a uma ou mais políticas.

Policy: Classe que possui informações que governam o uso de recursos da rede em um contexto particular e como diferentes recursos da rede interagem entre si.

LinkedContainer: Classe que implementa vínculos (links), permitindo que recipientes (containers) sejam arranjados como listas encadeadas.

Network-Media: Classe base para definição de como elementos de rede utilizam diferentes meios para se comunicar entre si e com outros componentes do sistema.

5. PRINCIPAIS CLASSES DO ESQUEMA PARA IMPLEMENTAÇÕES RELATIVAS A SEGURANÇA

Até aqui foram apresentadas várias classes de objetos definidas pelos modelos X.500, CIM e DEN que compoem o esquema global para o serviço de diretório definido pelo DMTF. Neste capítulo, será definido um esquema que é um subconjunto do esquema esquema global que fornecerá o suporte para as aplicações e serviços de segurança. Este esquema de segurança é composto de algumas classes da especificação DEN e classes definidas por este trabalho que estendem o DEN, permitindo assim que se implemente serviços de autenticação, autorização e políticas de segurança de forma integrada com o serviço de diretório.

Para facilitar o entendimento do esquema de segurança, ele foi subdividido em um sub esquemas para autenticação, um para autorização e outro para políticas. Para a aplicação do esquema de segurança podem ser utilizadas as diferentes arquiteturas de infra-estrutura apresentadas a seguir.

Arquiteturas para Aplicação do Serviço de Diretório

A arquitetura da infra-estrutura para dar suporte à proposta apresentada por este trabalho pode ser dividida em basicamente duas categorias. A primeira, presupõe que os dispositivos de rede estejam preparados para interagir com o diretório, ou seja são dispositivos habilitados ao serviço de diretório. Já a segunda, trata dispositivos de rede que não estão preparados para utilizar um serviço de diretório de forma integrada (dispositivos legados).

5.1.1. Arquitetura para Dispositivos Habilitados ao Serviço de Diretório

As arquiteturas analisadas para que os processos de autenticação, autorização e políticas de segurança sejam executados por dispositivos habilitados ao serviço de diretório (*directory-enabled)* são compostas pelos seguintes componentes:

Enforcer: Dispositivo que garante que uma regra é seguida. Um *enforcer* pode garantir ou restringir o uso. Seu papel é obrigar a complacência com a política a ser executada devido a requisição do recurso.

Interpreter: Dispositivo que avalia as circunstâncias da requisição de um recurso e entrega uma resposta para o *enforcer.*

O *interpreter* consulta várias fontes de informação antes de repassar sua decisão. Se estas fontes são raramente alteradas (bancos de dados e diretórios) são chamadas de estáticas. Se são alteradas constantemente, são chamadas de dinâmicas. A resposta que o *interpreter* envia para o *enforcer* é conhecida como *policy lease*.

Note que estes papéis são executados com a rede. É perfeitamente aceitável, e muitas vezes apropriado, que um dispositivo seja o requisitor, o *enforcer* e o *interpreter* ao mesmo tempo. No caso analisado aqui, é preciso que o dispositivo ou elemento requisitor seja diferente do *enforcer*, pois este deve garantir a segurança executando políticas de segurança e não poderia ser o mesmo mecanismo que envia a requisição. Desta forma, dois modelos podem ser implementados seguindo a arquitetura para dispositivos habilitados ao serviço de diretório.

No primeiro modelo, os papéis de *enforcer* e *interpreter* são executados por um mesmo dispositivo. Por exemplo, um *firewall* se adapta a este modelo pois ele buscará as informações ou definições de regras no diretório (papel do *interpreter*) e garantirá que o requisitor seguirá estas regras (papel do *enforcer*). Este modelo pode ser visualizado na figura 5.1 a seguir:

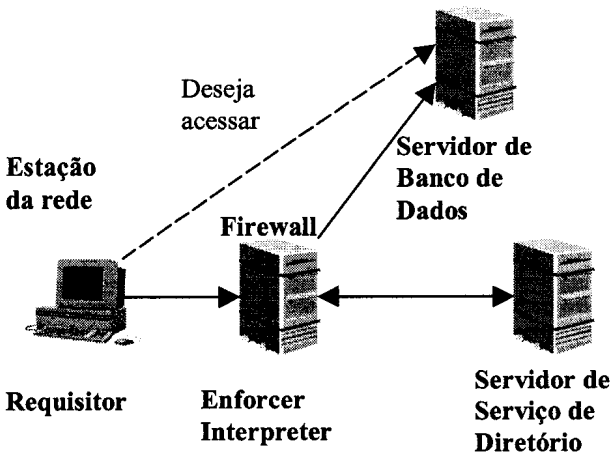

Figura 5.1 – Modelo de Enforcer e Interpreter Integrados

Na figura 5.1, o requisitor envia uma requisição de acesso a algum recurso (banco de dados) que está além do *firewall*. O *firewall*, fazendo o papel do *interpreter*, busca as informações de autorização de acesso no servidor de diretório e define se o acesso pode ser realizado ou não. Após a definição o próprio *firewall*, fazendo o papel de *enforcer* garante que a regra seja seguida (ou barra o acesso ou permite o acesso).

No segundo modelo, todos os componentes são desempenhados por dispositivos independentes. Este modelo é apresentado na figura 5.2 a seguir:

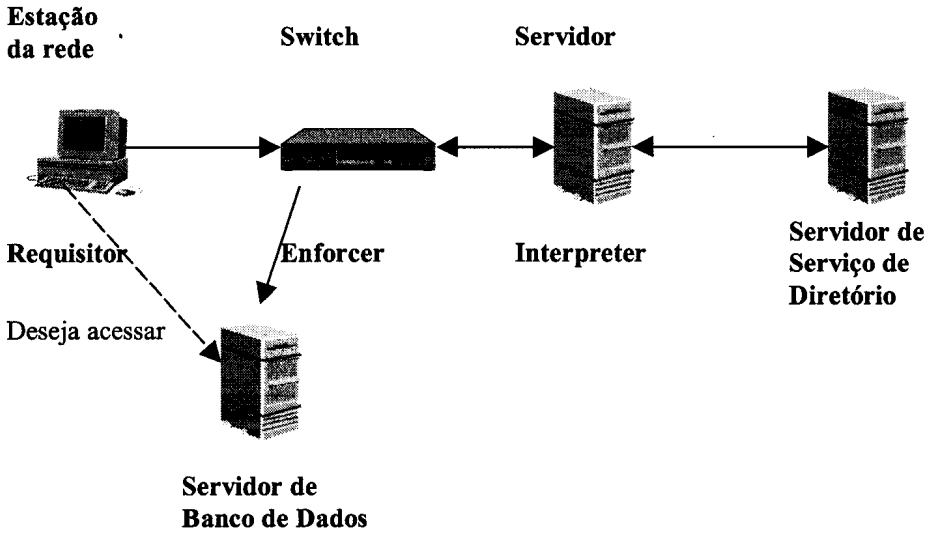

Figura 5.2 – Modelo com componentes independentes

Na figura 5.2 acima, o requisitor solicita um acesso a um recurso da rede (banco de dados). O *switch*, que no exemplo realiza o papel do *enforcer*, passa as informações de acesso para o servidor que faz o papel de *interpreter*. O *interpreter,* busca as informações no servidor de diretório e define se o acesso pode ser realizado ou não. Tendo esta definição o resultado é repassado para o *enforcer* que irá permitir ou não o acesso ao recurso.

O modelo de dispositivos com papéis independentes está sendo implementado por vários fornecedores de hardware onde o papel do *enforcer* é realizado por um de seus dispositivos.

A arquitetura com dispositivos habilitados ao serviço de diretório é uma tendência do mercado pois, vários fornecedores de hardware já estão disponibilizando equipamentos que se encaixam nesta categoria. Além disso, os principais sistemas operacionais de rede já possuem um serviço de diretório integrado permitindo que estas implementações possam ser realizadas.

5.1.2. Arquitetura para Dispositivos Legados

Os dispositivos legados, ou seja, aqueles que não estão preparados para interagir diretamente com um serviço de diretório, podem se beneficiar de alguns recursos propostos por este trabalho. A implementação que pode ser realizada é a configuração automática de dispositivos de rede. Já a implementação baseada em políticas, que permite a definição de políticas dentro de um diretório que são avaliadas/executadas pelo *interpreter/enforcer*, não poderá ser utilizada, pois os dispositivos deveriam estar integrados com o serviço de diretório. Mais informações sobre políticas serão apresentadas mais adiante neste capítulo.

Para se implementar a configuração automática, deve ser desenvolvido um componente que faça a conexão com o diretório e realize as configurações necessárias nos dispositivos legados. Desta forma, o papel de integração com o diretório será realizado por este componente que busca as informações cadastradas e configura adequadamente cada dispositivo legado do sistema.

O componente que realizada a busca da informação no diretório e configura os dispositivos, poderia ser implementado como um serviço de configuração em um servidor ou estação que centralize o serviço. A infra-estrutura é demonstrada na figura 5.3 abaixo:

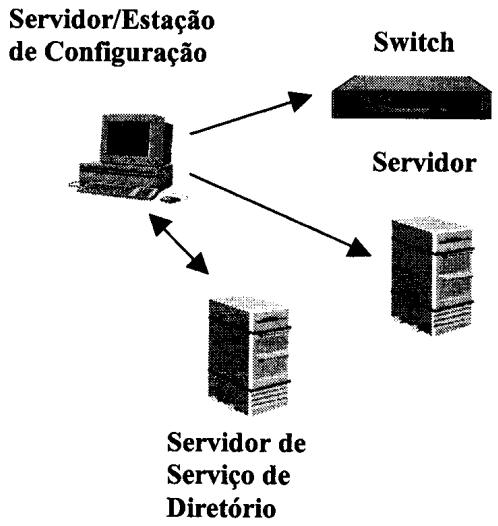

Figura 5.3 – Arquitetura para Dispositivos Legados

No exemplo da figura 5.3, uma aplicação de configuração é colocada no servidor de configurações. Esta aplicação, quando executada, busca informações de configuração no servidor de diretório e as aplica nos elementos de rede que necessitam ser configurados.

Um protótipo deste tipo de implementação foi construído e se encontra analisado no anexo 3.

Agora serão apresentados os sub esquemas que deverão ser aplicados para dar suporte a autenticação, autorização e políticas.

Aplicação da Autenticação

A autenticação é a prova da identidade, ou seja, provar quem você diz que é. Para que um usuário se conecte e tenha acesso a rede de computadores este deve se autenticar.

Para que um usuário seja autenticado informações de identificação (por exemplo, nome do usuário para o sistema e sua senha) são necessárias. Além disso, informações de local de onde o usuário se conecta e profiles do usuário podem ser necessárias dependendo da implementação do serviço de segurança que dá suporte a autenticação.

5.1.3. Eventos na Autenticação

Para que se entenda a os processos envolvidos na autenticação, abaixo é apresentada a sequência de eventos que ocorre durante a sua execução:

1. Um usuário requisita sua autenticação. A requisição é enviada para a aplicação de autenticação (papel *enforcer*)
2. O *enforcer* repassa a requisição para o *interpreter*
3. A aplicação de autenticação (papel *interpreter*) decide o que consultar e busca as informações (usuário e senha) no diretório.
4. O serviço de diretório entrega as informações requisitadas.
5. A aplicação de autenticação (papel *interpreter*) sintetiza as informações .
6. A aplicação de autenticação (papel enforcer) executa a autenticação.

A figura 5.4 abaixo mostra os passos apresentados anteriormente. O papel do *enforcer* e o *interpreter* são executados pela aplicação/serviço de autenticação.

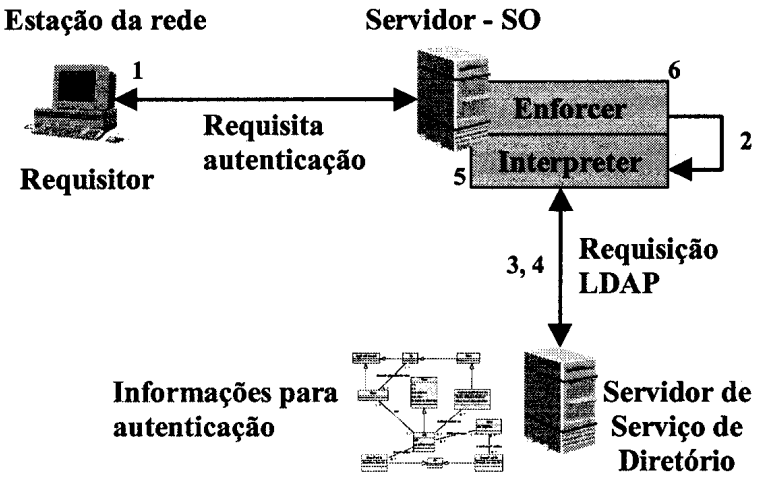

Figura 5.4 - Eventos para Autenticação

5.1.4. Modelo de Objetos

A seguir são apresentadas as classes necessárias para a aplicação da autenticação. Para isso, serão utilizadas classes definidas pela especificação DEN e algumas extensões realizadas através de relacionamentos e novas classes definidas por este trabalho. O diagrama de classes é mostrado a seguir na figura 5.5 (o diagrama apresenta apenas classes interessantes para aplicação da autenticação):

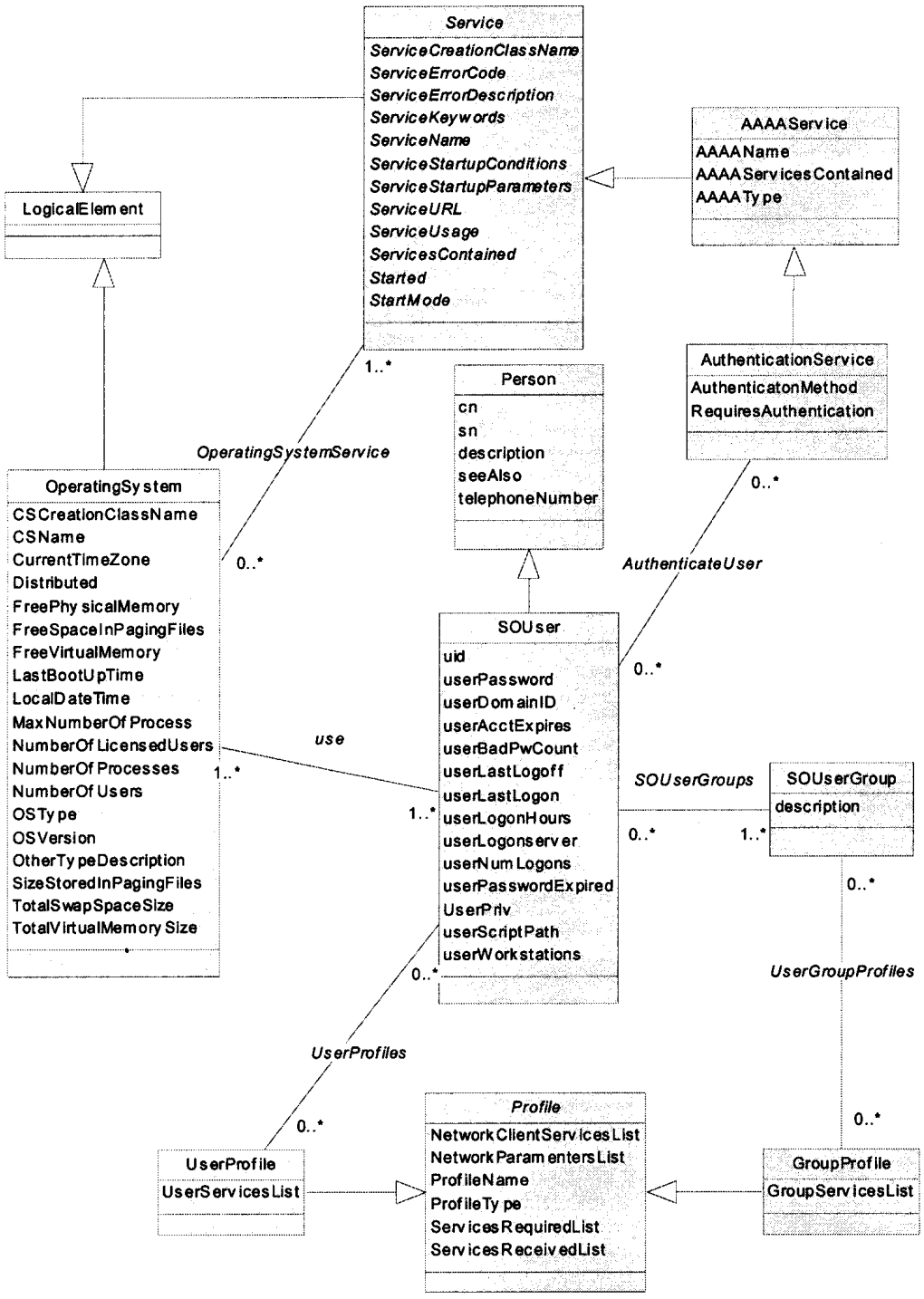

Figura 5.5 - Diagrama de Classes para Aplicação da Autenticação no Serviço de Diretório

O diagrama de classes apresentado pela figura 5.5, estrutura todas as informações necessárias para a aplicação da autenticação. Estas informações compreedem dados de todas as entidades envolvidas no processo de autenticação: usuário, sistema operacional, serviço de autenticação, profiles de usuários e grupos de usuários. Abaixo é apresentada um breve descrição das classes que compreendem este diagrama.

Descrições das Classes do Modelo

- LogicalElement

  Classe abstrata base para todos os componentes de um sistema que representa uma funcionalidade abstrata como: arquivos, processos, etc.
- OperatingSystem

  Representa um sistema operacional, definido como um software/firmware que faz com que um sistema computacional seja utilizável e implementa e/ou gerencia os recursos disponíveis neste sistema.
- Service

  Elemento lógico que contém informações necessárias para representar e gerenciar a funcionalidade por um dispositivo ou características de software.
- AAAAService

  Define objetos serviço que podem ser utilizados para implementar serviços de autorização, autenticação, contabilização e auditoria. É esperado que cada objeto AAAA será compreendido de um número de objetos que controlará um ou mais aspectos de todo o conjunto de serviços AAAA descritos pelo objeto AAAA pai.

- AuthenticationService

  Define objetos serviço que podem ser utilizados para prover autenticação de um dado objeto.

- Person

  Classe utilizada para definir o conceito genérico de pessoa.

- SOUser

  Classe utilizada para definir um susário cadastrado no sistema operacional.

- Profile

  Define características e necessidades de um objeto. Por exemplo, um objeto UserProfile poderia definir como um usuário conecta-se a rede (características) e quais serviços estão disponíveis para aquele usuário uma vez que ele esteja conectado.

- UserProfile

  Define os serviços que o usuário está autorizado a utilizar. Também define se o usuário necessita ser autenticado antes do serviço ser concedido e que método de autenticação deve ser utilizado

- SOUserGroup

  Identifica os grupos de usuários definidos no sistema operacional.

- GroupProfile

  Define os serviços que um grupo está autorizado a utilizar. Todo usuário que for membro deste grupo possui estes serviços por default.

Para uma descrição mais detalhada incluindo atributos veja o anexo 2.

Para a aplicação da autenticação, além das classes LogicalElement, Service, AAAService, AuthenticationService, OperatingSystem, Person, Profile,

UserProfile e GroupProfile descritas no DEN, foram adicionadas as classes SoUser e SoUserGroup para dar suporte aos usuários cadastrados no sistema operacional de rede. Desta forma, as características destes usuários podem ser armazenadas no serviço de diretório. Além destas duas classes, para descrever a relação entres estas classes, este trabalho acresenta os seguintes relacionamentos: AuthenticateUser, que define qual serviço de autenticação disponível aquele usuário deve receber; SOUserGroups, que define quais os grupos de usuários o usuário pertence; use, que define quais sistemas operacionais este usuário pertence; UserProfiles, que define os profiles que este usuário deve obedecer; UserGroupProfiles, que define quais os profiles que os grupos de usuários, ao qual o usuário pertence, deve obedecer.

5.1.5. Modelo Dinâmico

Para mostrar a interação entre a aplicação de autenticação, mais especificamente do *interpreter*, e os objetos definidos no diagrama de classes da figura 5.5, foi elaborado um diagrama de seqüência. Esta interação é descrita abaixo:

Interações para Autenticação

Após o usuário requisitar a conexão, a aplicação de segurança através de seu *interpreter* interage com o diretório da seguinte forma:

1. Busca as informações do usuário na classe SOUser;
2. Através da classe AuthenticationService verifica qual o tipo de autenticação deve ser utilizada para este usuário através do atributo AuthenticationMethod;
3. Verifica as informações de grupos de usuários aos quais o usuário pertença acessando a classe SOUserGroup;

4. Caso seja implementado, verifica os profiles definidos para os grupos do usuário;
5. Verifica os profiles definidos para o usuário especificamente.

Estes passos podem ser verificados no diagrama de sequência da figura 5.6.

Para verificar como deve ser interpretado um diagrama de sequência veja o anexo 4.

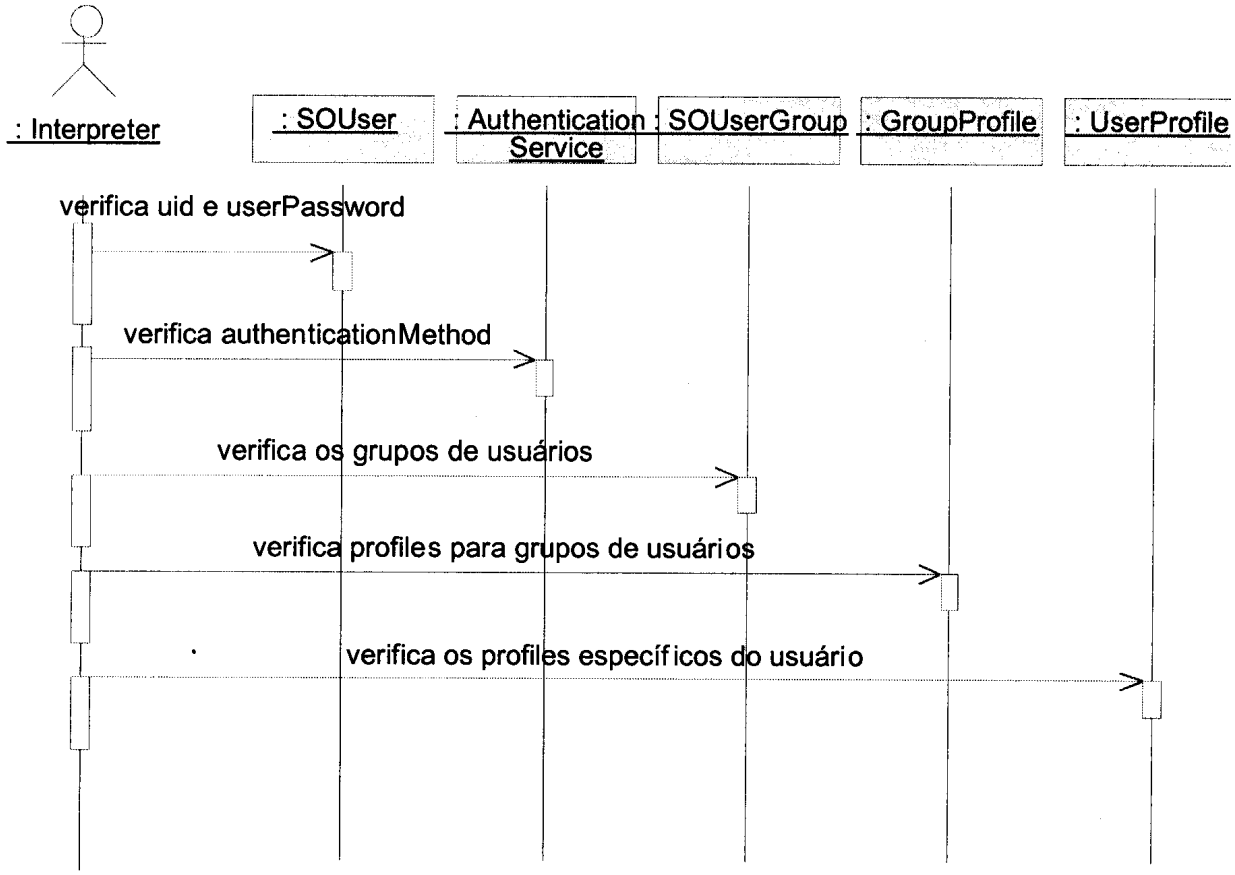

Figura 5.6 - Diagrama de Seqüência para Autenticação

A aplicação ou serviço que desempenhar o papel do *interpreter* deverá interagir com o diretório como apresentado na figura 5.6 e após a busca das informações deverá tratá-las de forma a definir a resposta a ser dada ao enforcer, ou seja, permitir a conexão ou não.

Aplicação da Autorização

Quando um usuário acessa um recurso, existe a necessidade de identificar quais os direitos de acesso ele possui, ou seja, que privilégios este usuário possui em relação ao recurso a ser acessado. Esta fase é conhecida como autorização.

Para a autorização, todo objeto que for definido dentro do repositório do diretório que necessite de permissões de acesso deve possuir um relacionamento com a classe ACL (Access Control List) [BRAIN96]. Uma ACL é uma lista de controle de acesso composta por ACEs (Access Control Entry) [BRAIN96] que são entradas que determinam quem pode acessar o objeto e o que se pode fazer com o objeto. Uma ACE é composta por três partes: um identificador de usuário ou grupo (SID), um ACE header que determina o tipo de acesso (*access allowed* - acesso permitido, ou *access denied* – acesso não permitido) e um ACE mask que determina o que o usuário pode fazer com o objeto (*read* - leitura, *write* - escrita, *read-write*).

5.1.6. Eventos da Autorização

Para que se entenda os processos envolvidos na autorização, abaixo é apresentada a sequência de eventos que ocorre durante a sua execução:

1. Um usuário/processo requisita acesso a um objeto que necessita de autorização. A requisição é enviada para a aplicação que realiza o papel do *enforcer* para a autorização (sistema operacional, banco de dados, etc);
2. O *enforcer* repassa a requisição para o *interpreter*;
3. A aplicação de autorização, através de seu *interpreter*, requisita a ACL e ACEs do objeto para o servidor de diretório (papel *interpreter*).
4. O servidor de diretório entrega as informações requisitadas.

5. A aplicação de autorização (papel *interpreter*) sintetiza as informações.
6. A aplicação de autorização (papel *enforcer*) executa a autorização.

A figura 5.7 abaixo mostra os passos apresentados anteriormente. O papel do *enforcer* e o *interpreter* são executados pela aplicação/serviço de autorização.

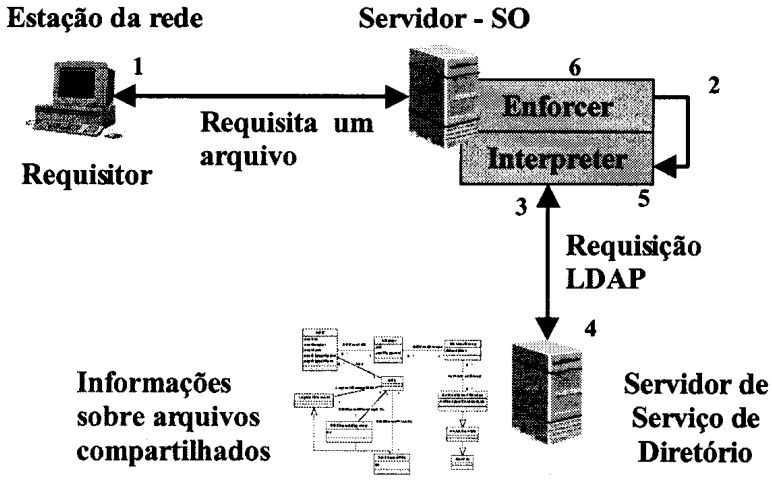

Figura 5.7 - Eventos para Autorização

5.1.7. Modelo de Objetos

A seguir, é apresentado o diagrama de classes que deve ser utilizado para realizar a autorização baseada no serviço de diretório. Todos os objetos lógicos que necessitem de autorização devem possuir um relacionamento com a classe ACL, isto pode ser verificado através do relacionamento LogicalElementACL, mostrado no diagrama de classes da figura 5.8 abaixo.

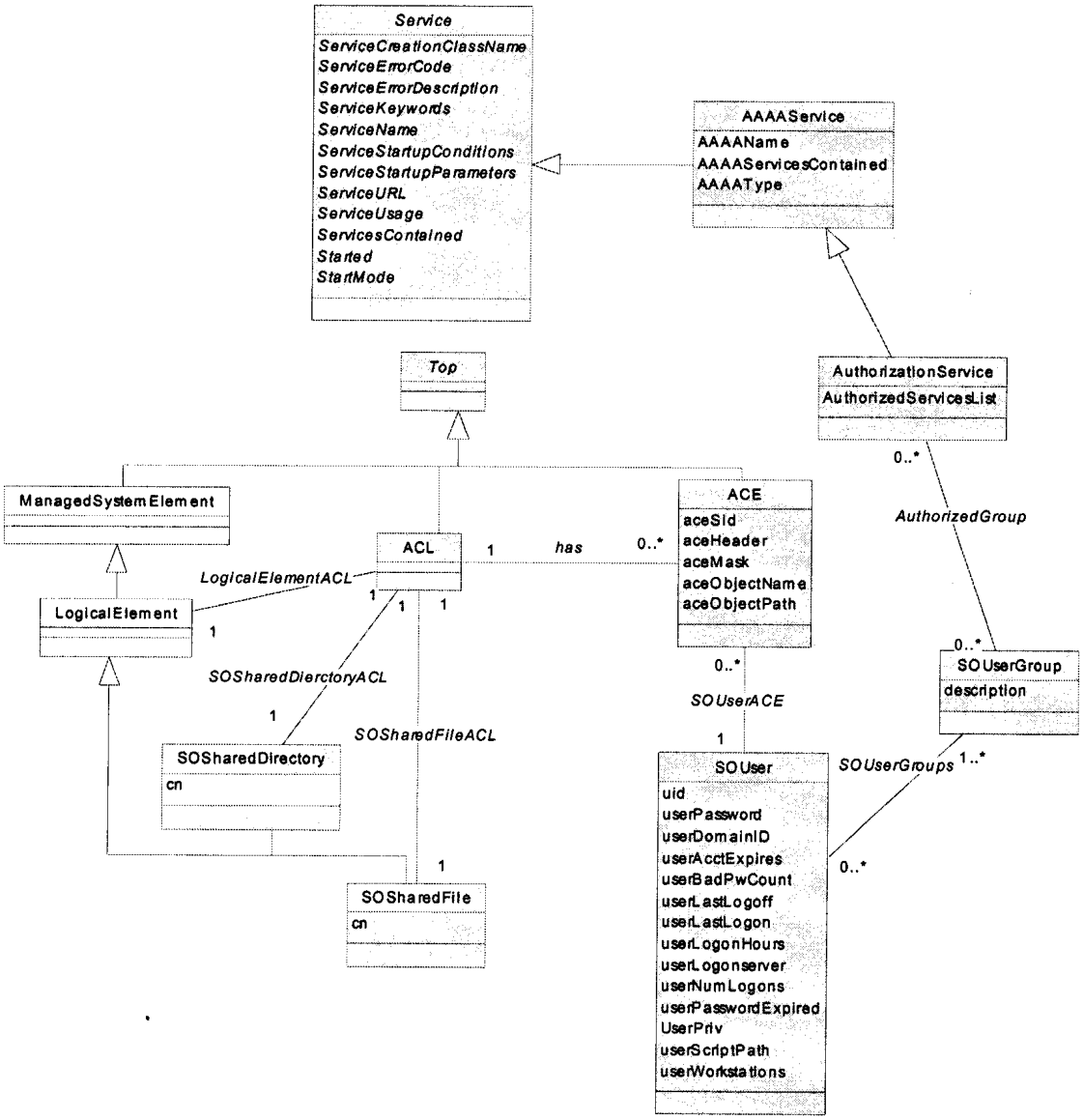

Figura 5.8 - Diagrama de Classes para Aplicação da Autorização no Serviço de Diretório

O diagrama de classes apresentado pela figura 5.8, estrutura todas as informações necessárias para a aplicação da autorização. Estas informações compreendem dados de todas as entidades envolvidas no processo de autorização como: usuários e seus grupos, serviços de autenticação, elementos que devem sofrer autorização e as listas de permissões.

Para a aplicação da autorização, além das classes descritas no DEN, este trabalho adiciona as classes SoUser, SoUserGroup, SOSharedDirectory e

SOSharedFile para agruparem os usuários cadastrados no sistema operacional de rede e os arquivos e diretórios compartilhados na rede, pois apenas os recursos compartilhados (no caso arquivos e diretórios) devem ser armazenados no diretório global. Além destas, outras duas classes são necessárias para a aplicação da autorização: ACE e ACL. Através destas duas classes, para qualquer recurso que se deseje pode-se armazenar de forma centralizada as permissões de acesso aos recursos.

No diagrama de classes acima, as classes SOSharedFile (classe para arquivos compartilhados) e SOSharedDirectory (classes para diretório compartilhados) herdam o relacionamento LogicalElementACL representados por SOSharedFileACL e SOSharedDirectoryACL para relacioná-las com a classe ACL.

Para qualquer recurso compartilhado que se deseje armazenar suas permissões (ACLs) de forma centralizada, basta definir uma subclasses da classes DEN LogicalElement e relacioná-la com a classe ACL. A seguir é apresentada uma breve descrição das classes que aparecem no diagrama da figura 5.8.

Descrições das Classes do Modelo

- LogicalElement

  Classe abstrata base para todos os componentes de um sistema que representa uma funcionalidade abstrata como: arquivos, processos,ou outros elementos.

- Service

  Elemento lógico que contém informações necessárias para representar e gerenciar a funcionalidade por um dispositivo ou características de software.

- AAAAService

  Define objetos do tipo serviço que podem ser utilizados para implementar serviços de autorização, autenticação, contabilização e auditoria. É esperado que cada objeto AAAA será compreendido de um número de objetos que controlará um ou mais aspectos de todo o conjunto de serviços AAAA descritos pelo objeto AAAA pai.

- SOUser

  Classe utilizada para definir um usuário cadastrado no sistema operacional.

- Profile

  Define características e necessidades de objetos. Por exemplo, um objeto UserProfile pode definir como um usuário conecta-se a rede (características) e quais serviços estão disponíveis para aquele usuário uma vez que ele esteja conectado.

- Top

  Classe Origem do diretório da qual todas as outras classes são derivadas. Esta classe foi buscada no esquema do X.500 e estendida para modelar elementos de rede.

- AuthorizationService

  Classe que define objetos serviço que podem ser utilizados para fornecer autorização para um dado objeto.

- ACL

  Classe que identifica as ACLs dos objetos que devem sofrer autorização.

- ACE

  Classe que identicfica as ACEs de uma dada ACL.

- SOUserGroup

  Identifica os grupos de usuários definidos no sistema operacional.

- SOSharedFile

  Identifica os arquivos compartilhados no sistema operacional. Está classe é implementada para que o diretório contenha apenas os dados dos arquivos compartilhados, pois apenas estes arquivos podem sofrer autorização.

- SOSharedDirectory

  Identifica os diretórios compartilhados na rede. Esta classe é implementada para que o diretório contenha apenas os dados dos diretórios compartilhados, pois apenas estes devem sofrer autorização.

Para uma descrição mais detalhada incluindo atributos veja o anexo 2.

5.1.8. Modelo Dinâmico

Para mostrar a interação entre a aplicação de autorização, mais especificamente do *interpreter*, e os objetos definidos no diagrama de classes da figura 5.8, foi elaborado um diagrama de seqüência. Esta interação é descrita abaixo:

Interações para Autorização

Após o usuário requisitar a conexão, a aplicação de segurança através de seu *interpreter* interage com o diretório da seguinte forma (estes passos podem ser verificados no diagrama de sequência da figura 5.9):

1. O *interpreter* verifica as informações do objeto que deve sofrer autorização. No caso do exemplo da figura 5.9, este objeto é um diretório compartilhado (diretório do sistema operacional) que é definido pela classe SOSharedDirectory
2. O *interpreter* busca a ACL do objeto que deve sofrer autorização

3. Através da ACL selecionada, o interpreter busca as ACEs cadastradas para aquela ACL.

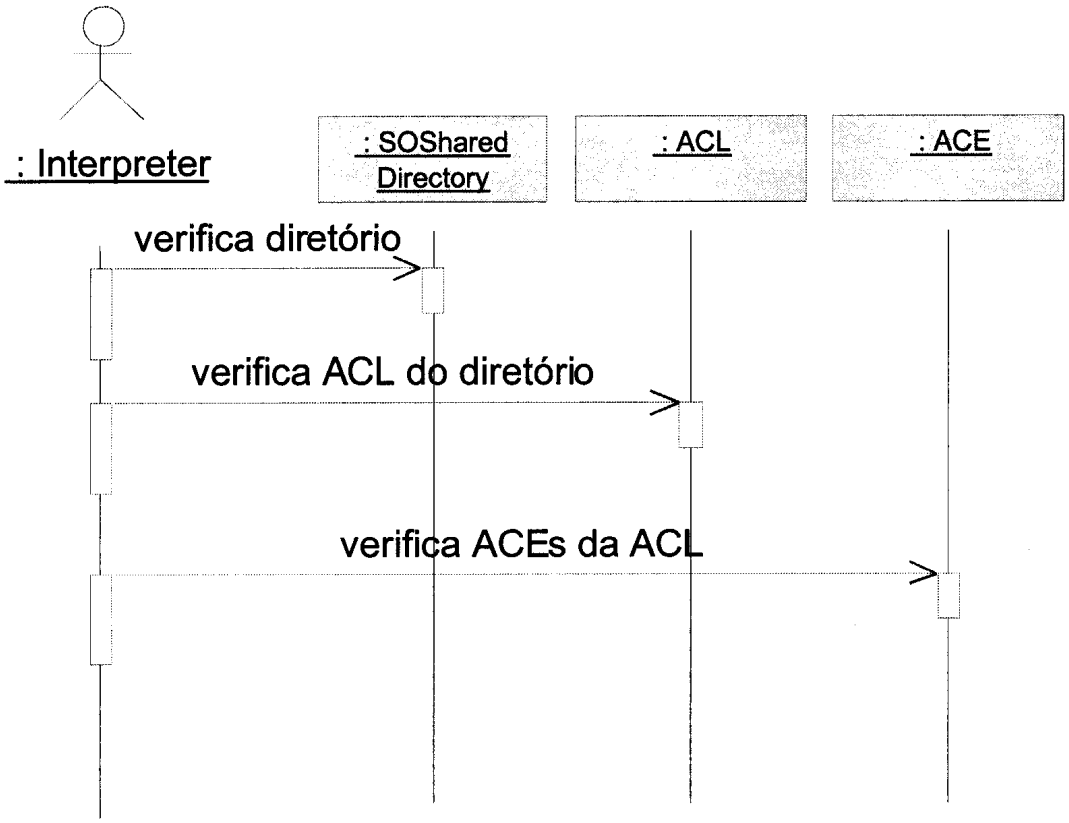

Figura 5.9 - Diagrama de Seqüência para Autorização

Com a aplicação dentro do diretório do sub esquema de autorização juntamente com o sub esquema de autenticação apresentado anteriormente, alcança-se um dos objetivos propostos por este trabalho que é a aplicação da base de dados de autenticação e autorização centralizada logicamente em um único ponto.

Com isso, a administração da segurança básica, acesso a um recurso e permissão do que se pode fazer com o recurso, será facilitada.

Aplicação de Políticas de Segurança

Um sistema de segurança que possa atender as necessidades dos ambientes atuais, deve possuir dentro do repositório do diretório um esquema que dê suporte a implementação de políticas de segurança.

Políticas podem ser definidas como sendo um conjunto de regras que governam a alocação de recursos. Uma transação de uma política inicia com a requisição de um recurso que pode ser por exemplo o início de um fluxo a ser roteado, o acesso a uma porta ethernet, etc.

A adoção de um esquema para políticas, quando integrada com serviços/aplicações de segurança como firewall, scanners, sniffers, etc, vai permitir a definição de regras de segurança integradas com o ambiente das das empresas. Desta forma, poderá ser adotada uma segurança altamente adaptativa, ou seja, de acordo com as características que o ambiente possui em relação ao tempo, configurações de segurança poderão ser realizadas automaticamente para garantir a integridade do ambiente, momento a momento. Para maiores informações sobre ferramentas de segurança veja [GON97] e [ZVER00]. O primeiro passo adoção deste tipo de serviço é a definição do esquema de diretório. Após esta definição, deve ser feita a integração dos serviços/aplicações de segurança com o diretório.

5.1.9. Eventos para Aplicação de Políticas

Para que se entenda os processos envolvidos na requisição de uma política, abaixo é apresentada a sequência de eventos que ocorre durante a sua execução:

1. Um recurso requisita utilização de outro recurso. A requisição é enviada para o componente (aplicação ou dispositivo) que atua como *enforcer*
2. O *enforcer* requisita as informações necessárias para o *interpreter*

3. O *interpreter* requisita uma variedade de informações de fontes de dados estáticos e dinâmicos
4. As fontes de dados (agentes SNMP, servidor de diretório, etc) disponibilizam as informações para o *interpreter*
5. *O interpreter* formula uma resposta em forma de uma permissão
6. A permissão é passada para o *enforcer* que aplica esta permissão para o dispositivo requisitor para garantir que o uso da rede segue o conjunto de diretrizes especificadas na política.

A figura 5.10 abaixo apresenta esta sequência de passos:

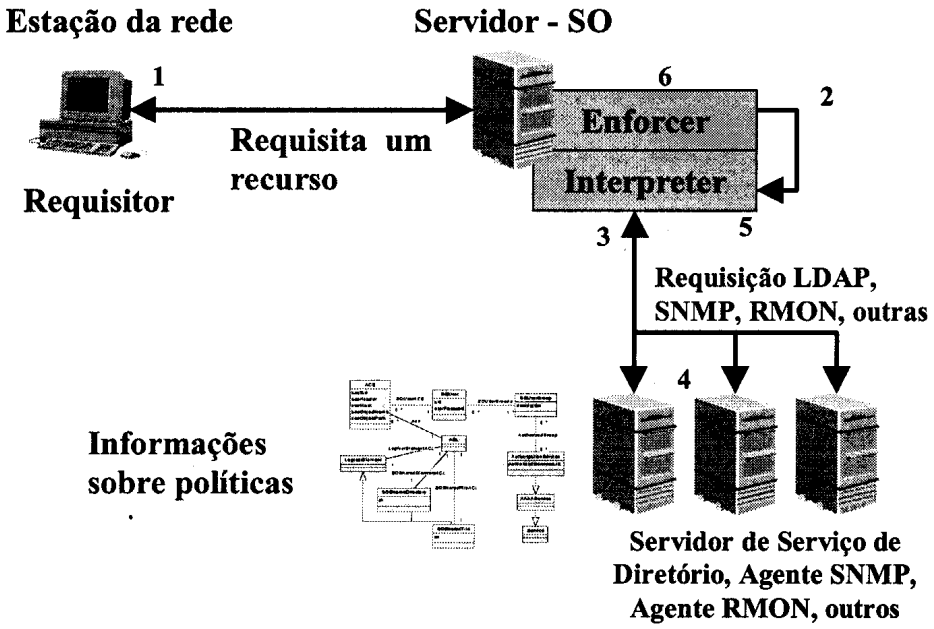

Figura 5.10 - Eventos para Políticas

A figura 5.10, mostra que a aplicação de políticas pode buscar informações em outras fontes de dados como RMON, SNMP, etc. Para maiores informações sobre estas fontes e gerenciamento de rede veja [HAR96], [STA96], [RFC1155], [RFC1157], [RFC1212], [RFC1215] e [TAN96].

5.1.10. *Modelo de Objetos*

A figura 5.11 apresenta as classes necessárias para a aplicação de políticas. Para isso, serão utilizadas classes definidas pela especificação DEN, e algumas extensões realizadas por este trabalho através da criação de relacionamentos e novas classes.

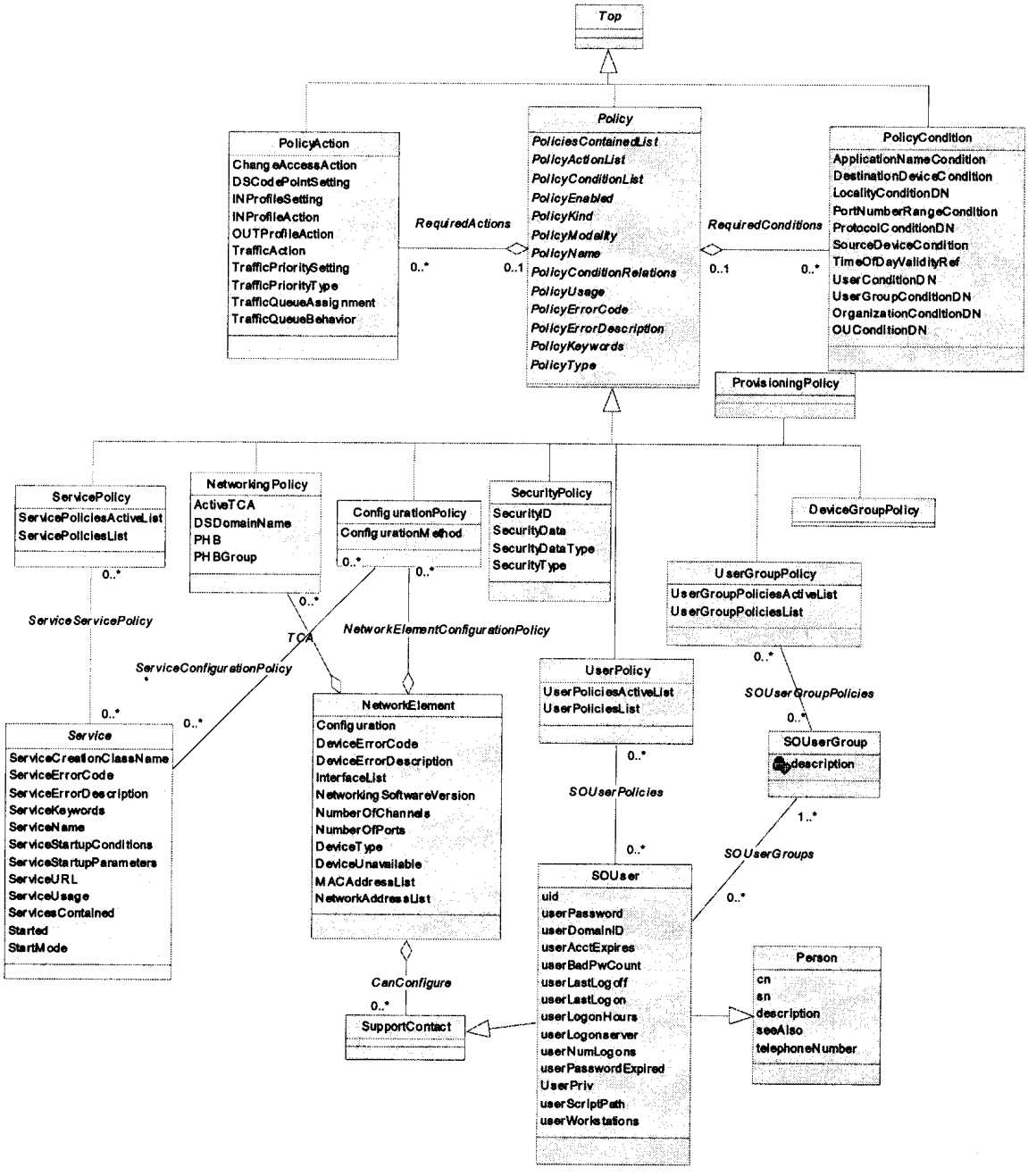

Figura 5.11 - Diagrama de Classes para Aplicação de Políticas no Serviço de Diretório

O diagrama de classes apresentado na figura 5.11, estrutura todas as informações necessárias para a aplicação de políticas. Aplicando-se este modelo, poderão ser definidas as políticas (classe Policy e derivadas), suas condições (classe PolicyCondition) e as ações (classe PolicyAction) a serem executadas de acordo com as condições estabelecidas. Com isso, será possível a implementação de um ambiente de rede onde todos os elementos envolvidos (serviços de rede, aplicações, etc) poderão interagir para que em conjunto permitam a definição de regras que governem as necessidades específicas para o negócio.

Por exemplo, suponha que em um determinado ambiente deseja-se definir que após às 22:00 horas a aplicação do sistema financeiro, caso esteja sendo utilizada por algum usuário da filial 1 ou filial2 ou filial 3, não possa acessar o banco de dados do servidor SRVFINANCDB que responde na porta TCP 1521. Esta política poderia ser definida no sub esquema de políticas como mostrado na figura 5.12.

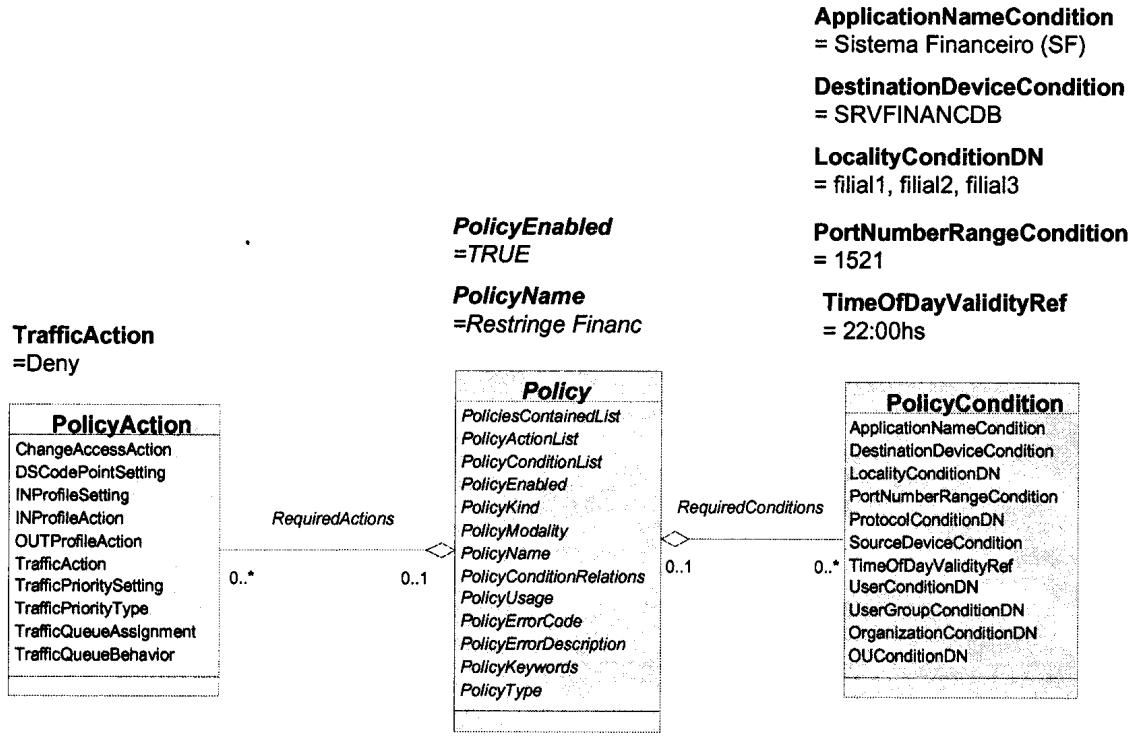

Figura 5.12 – Exemplo de Aplicação de uma Política

Pela classe Policy, no atributo PolicyName, define-se um nome para a política e no atributo PolicyEnable define-se que esta política está ativa. Pela classe PolicyCondition define-se as condições da política pelos seguintes atributos: ApplicationNameCondition define a aplicação que respeitará esta política, DestinationDeviceCondition define o dispositivo que receberá a condição, LocationConditionDN define os locais que devem respeitar esta condição, PortNumberRangeCondition define a porta que deve respeitar a condição, TimeOfDayValidityRef define a hora que a condição será aplicada. Pela classe PolicyAction define-se que o tráfego deve ser "barrado" caso a condição seja estabelecida.

Abaixo segue uma breve descrição das classes apresentadas no diagrama da figura 5.11.

Descrições das Classes do Modelo

- Service

  Elemento lógico que contém informações necessárias para representar e gerenciar a funcionalidade por um dispositivo ou características de software.
- Person

  Classe utilizada para definir o conceito genérico de pessoa.
- SOUser

  Classe utilizada para definir um susário cadastrado no sistema operacional.
- Profile

  Define características e necessidades de um objeto. Por exemplo, um objeto UserProfile poderia definir como um usuário conecta-se a rede (características) e quais serviços estão disponíveis para aquele usuário uma vez que ele esteja conectado.

- Top

  Classe Origem do diretório da qual todas as outras classes são derivadas. Esta classe foi buscada no esquema do X.500 e estendida para modelar elementos de rede.

- Policy

  Define um template para atributos e métodos que descrevem como funções podem ser invocadas para controlar como várias entidades interagem entre si.

- PolicyAction

  Classe utilizada para descrever o conjunto de ações a serem invocadas quando uma política é satisfeita. Se uma política requer múltiplas ações a serem executadas, então ela agregará vários objetos PolicyAction.

- PolicyCondition

  Classe utilizada para descrever o conjunto de condições que devem ser obedecidas para satisfazer uma política. Se uma política requer várias condições, então ela deve agregar vários objetos PolicyConditions.

- NetworkingPolicy

  Classe utilizada para se definir várias políticas de rede (Ex.: Como condicionar o tráfego de acordo com um conjunto de comportamentos definidos).

- SecurityPolicy

  Classe utilizada para definir várias políticas de segurança (Ex. IPsec)

- UserGroupPolicy

  Classe utilizada para definir políticas baseadas em usuários que usam um grupo para agregar múltiplos usuários.

- ConfigurationPolicy

  Classe utilizada para se definir vários meios de se configurar dispositivos.

- DeviceGroupPolicy

  Classe utilizada para definir políticas baseadas em grupo de dispositivos. Cada dispositivo pertencente ao grupo recebe esta política por default.
- UserPolicy

  Classe utilizada para se definir várias políticas baseadas em usuários.
- NetworkElement

  Representa características físicas de dispositivos de rede.
- SupportContact

  Pessoa responsável pelo suporte.

Para uma descrição mais detalhada veja o anexo 2.

5.1.11. Modelo Dinâmico

Para mostrar a interação entre o interpreter da aplicação ou serviço que implementa a política e os objetos definidos no diagrama de classes da figura 5.11, foi elaborado um diagrama de seqüência. Esta interação é descrita abaixo:

Interações para Políticas de Segurança

Após o usuário ou serviço requisitar algum serviço/recurso, a aplicação que implementa a política, através de seu *interpreter,* interage com o diretório da seguinte forma (estes passos podem ser verificados no diagrama de sequência da figura 5.13):

Os passos apresentados aqui estão definidos para usuários.

1. O interpreter busca as políticas definidas para o usuário através do atributo *UserPoliciesActiveList* definido na classe *UserPolicy*
2. Verifica as condições definidas para as políticas na classe *PolicyCondition*

3. Verifica as ações definidas para as políticas na classe *PolicyAction*
4. Verifica quais a quais grupos o usuário pertence na classe *UserGroup*
5. Verifica todas as políticas definidas para os grupos aos quais o usuário pertence na classe *UserGroupPolicy*
6. Verifica as condições definidas para as políticas dos grupos do usuário na classe *PolicyCondition*
7. Verifica as ações definidas para as políticas dos grupos do usuário na classe *PolicyAction.*

Para verificar como deve ser interpretado um diagrama de sequência veja o anexo4.

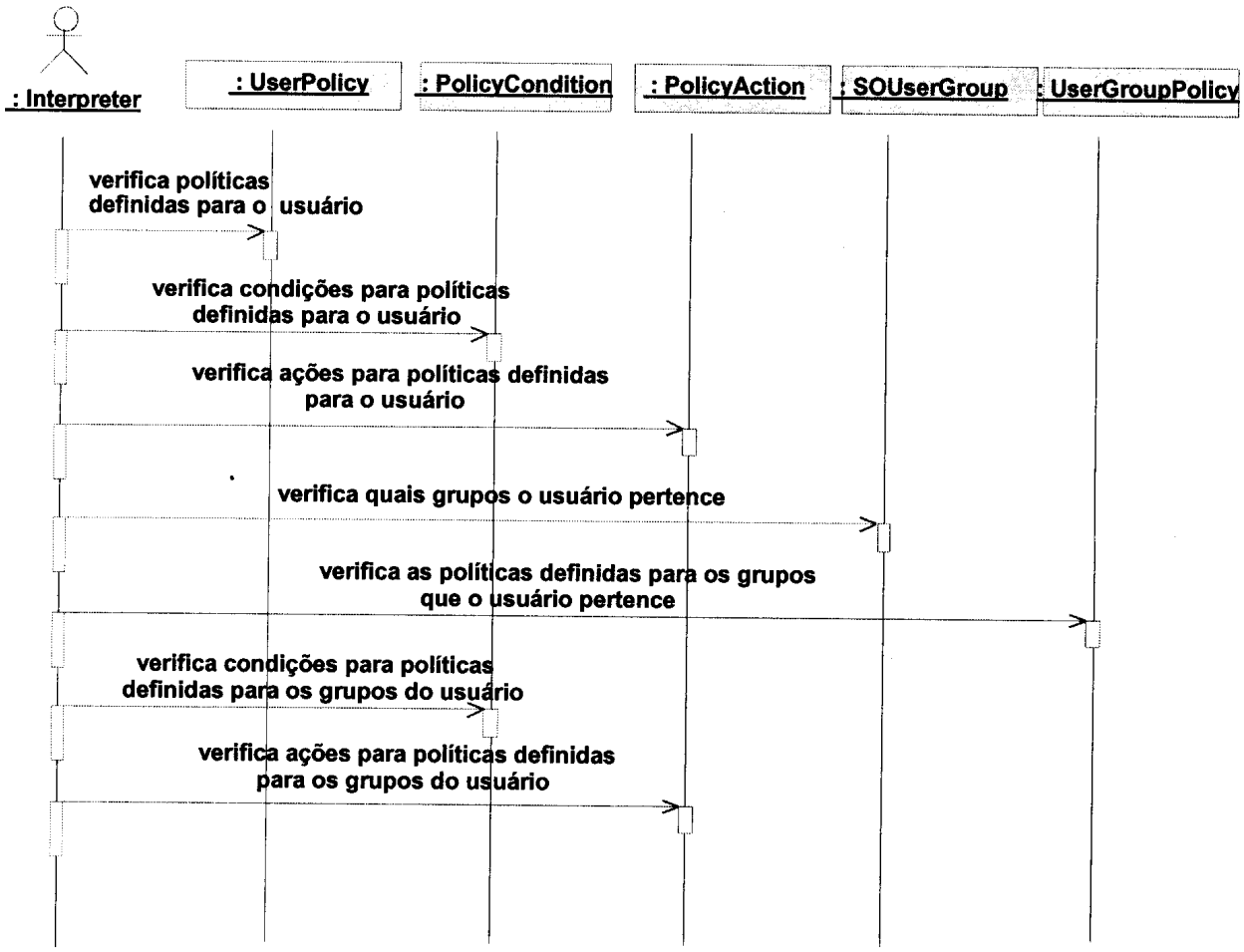

Figura 5.13 – Diagrama de Sequência para Políticas Aplicadas a Usuários

Para qualquer elemento de rede que se deseje implementar políticas basta estender o diagrama de classes da figura 5.11 criando uma subclasse para a classe Policy e criando os relacionamentos apropriados para estabelecer as relações entre os objetos. Por exemplo, no diagrama de classes da figura 5.11, além de políticas para usuário e serviços, está definido a classe NetworkingPolicy que permite a definição de políticas para elementos de rede como roteadores, switches, etc.

Com a aplicação deste sub esquema com os sub esquemas de autenticação e autorização, as aplicações/recursos que implementam os serviços de segurança poderão interagir entre si, permitindo que o ambiente possa se comportar de acordo com as suas necessidades, adaptando-se às condições momento a momento. Caso a rede a ser suportada pelo serviço de diretório esteja distribuída fisicamente em vários locais, é necessária a adoção de técnicas que permitam melhorar o desempenho e a disponibilidade. As técnicas mais utilizadas nestes casos são a distribuição e replicação de dados, apresentadas no próximo capítulo.

Estudo de caso: Ambiente da Cimento Rio Branco S/A

Para ilustrar os conceitos analisados, agora será demonstrada como seria, em termos gerais, a aplicação de um serviço de diretório para autenticação, autorização e políticas de segurança em um ambiente de uma grande empresa com várias unidades situados pelo Brasil. As características desta aplicação podem ser espelhadas para outras implementações maiores como por exemplo a Internet.

Cenário

A Cimento Rio Branco S.A. é uma empresa que faz parte do grupo Votorantim Cimento. A estrutura de informática da empresa está dividida em cinco unidades principais (Centro Administrativo, Fábrica Rio Branco, Itajaí, Esteio e Pinheiro Machado) e várias filiais. Para o estudo de caso será montado um esquema de aplicação para as cinco principais unidades. A figura 5.14 demonstra a estrutura que será analisada.

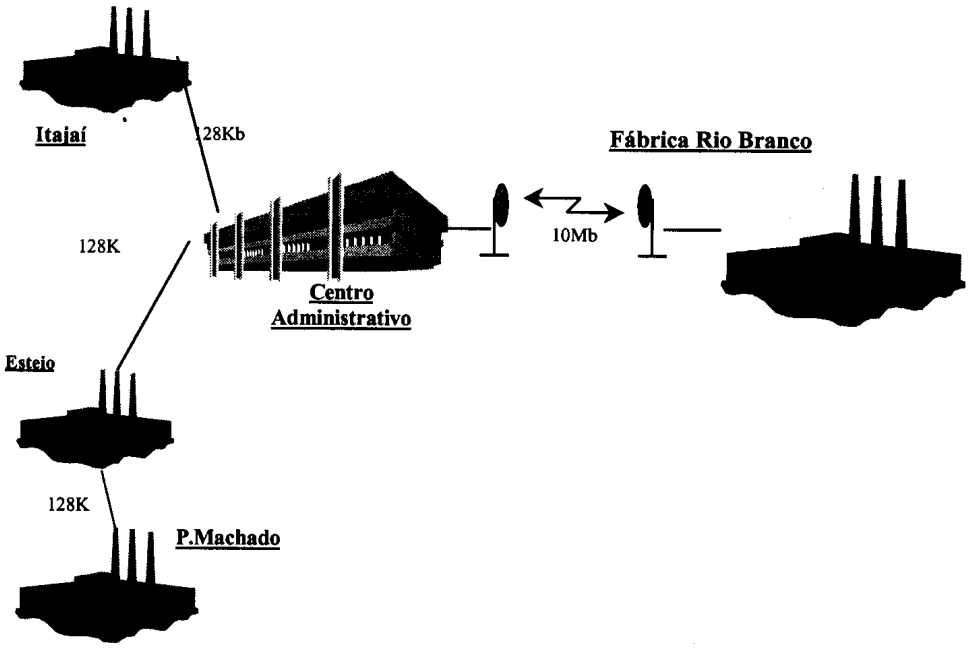

Figura 5.14 - Distribuição das Unidades da Cimento Rio Branco SA

Cada uma das unidades possui uma grande quantidade de estações e pelo menos um servidor para autenticação (controlador de domínio) e um ou mais servidores de aplicação (Banco de Dados e Mensagens).

Os principais problemas encontrados no ambiente são relacionados à administração das contas de usuários para autenticação, pois são muitos usuários e estes estão distribuídos pelas regiões e viajam muito entre as unidades e devem ser autenticados em qualquer ponto da rede; administração da autorização, que como visto anteriormente em ambientes de domínios fica local em cada nó, dificultando o controle das permissões. Esses problemas derivam do fato de não existir a possibilidade de se implementar políticas de segurança de forma integrada com a rede.

Modelo de Objetos

Para cada unidade deve ser alocado um servidor de diretório. Veja figura 5.15 abaixo:

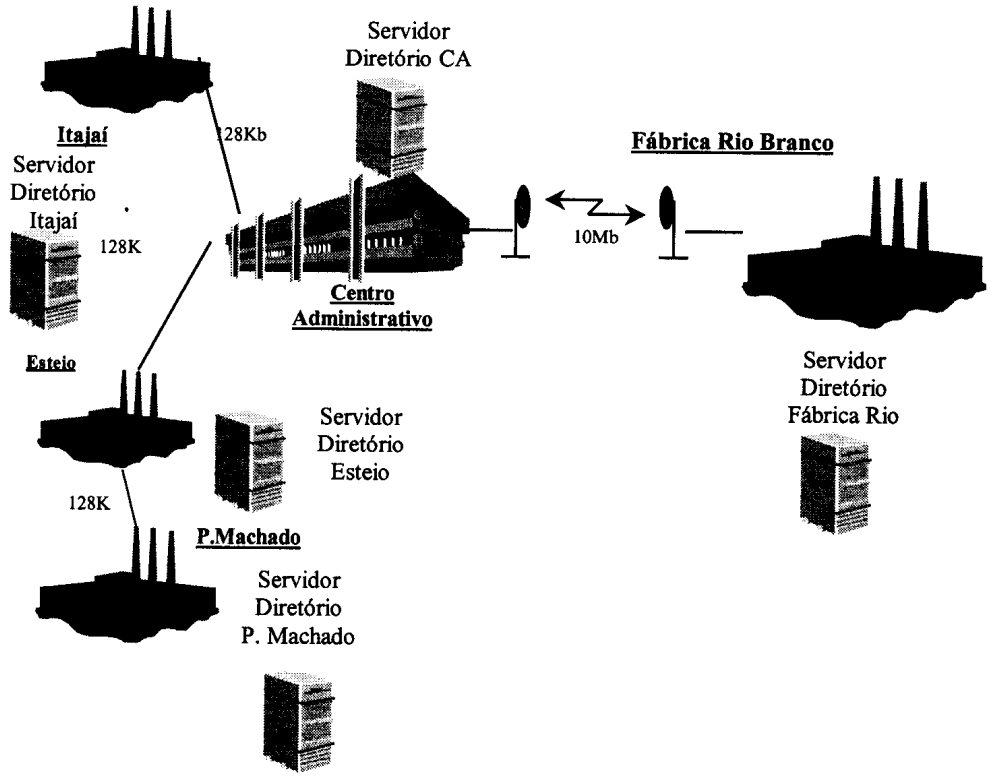

Figura 5.15 - Implementação de Servidores de Diretório nos diferentes Sites

Os modelos analisados anteriormente para autenticação, autorização e políticas devem ser implementados em cada servidor de diretório apresentado na figura 5.15. Além do esquema (base) todos os dados de segurança referentes a alguns usuários devem ser replicados, pois estes usuários se locomovem entre as unidades. Maiores informações sobre replicação do diretório podem ser verificadas no capítulo 6.

Para isto, deve ser implementado um grupo de usuários especial que deverá ser replicado. Para este grupo de usuários, será criada uma classe chamada *SOUserGroupEspecial* como subclasse da classe *SOUserGroup*. Os dados desta classe devem ser replicados para todos as demais unidades juntamente com seus usuários locais. A figura 5.16 apresenta esta nova classe juntamento com as classes apresentadas anteriormente na figura 5.5 que dão suporte a autenticação.

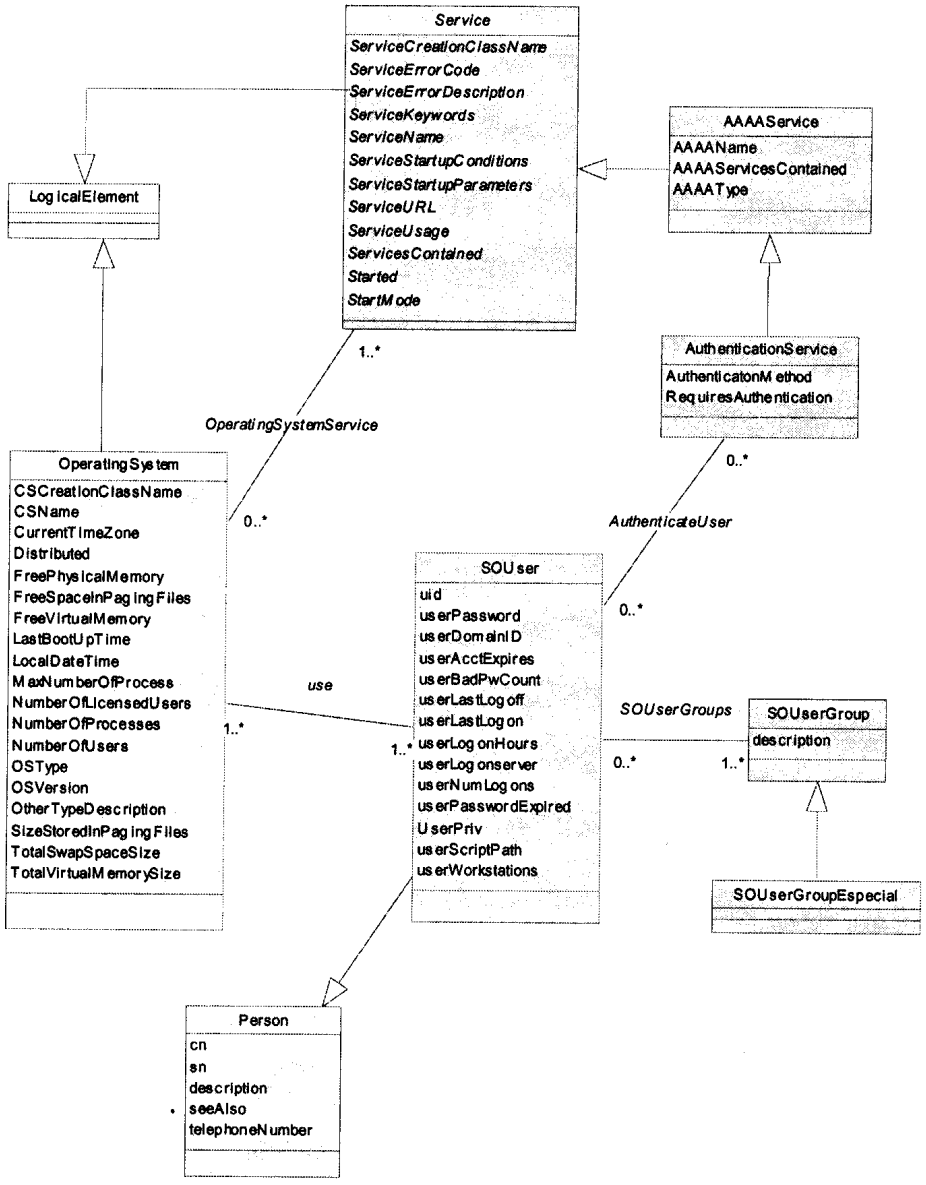

Figura 5.16 – Diagrama de Classes de Autenticação para o Ambiente Cimento Rio Branco S/A

A figura 5.17 mostra o diagrama que dá suporte a autorização. As informações de autorização (ACL e ACEs) de determinados diretórios e arquivos também devem ser replicadas pois vários destes recursos estão situados no CA (Centro Administrativo) e são compartilhados pelos usuários da rede corporativa.

Uma sub árvore com estes diretórios deve ser criada para ser replicada.

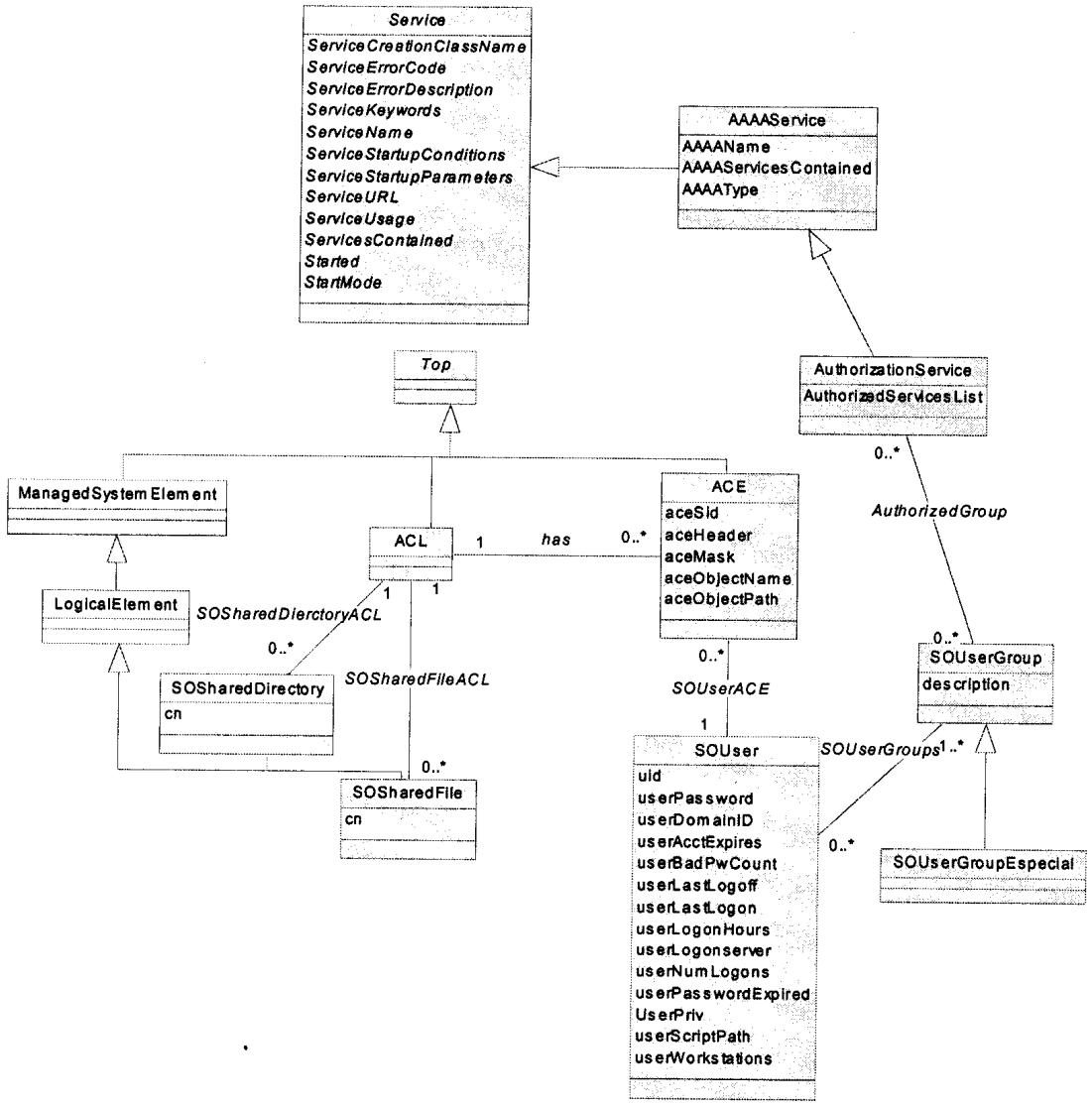

Figura 5.17 – Diagrama de Classes de Autorização para o Ambiente da Cimento Rio Branco S/A

Todos os arquivos e diretórios que são compartilhados na rede devem ser cadastrados no diretório, juntamente com sua ACL e ACEs, nas classes *SOSharedFile, SOSharedDirectory, ACL e ACE* apresentados no diagrama 5.17. Todos os dados destas classes devem ser replicados para todas as unidades para otimizar o acesso às informações.

Os dados de políticas que se referem aos usuário do grupo especial e dos recursos compartilhados também devem estar em todas as unidades. O modelo a ser utilizado é apresentado na figura 5.18. Este modelo contém todas as classes da figura 5.11 mais a classe *SOUserGroupEspecial*.

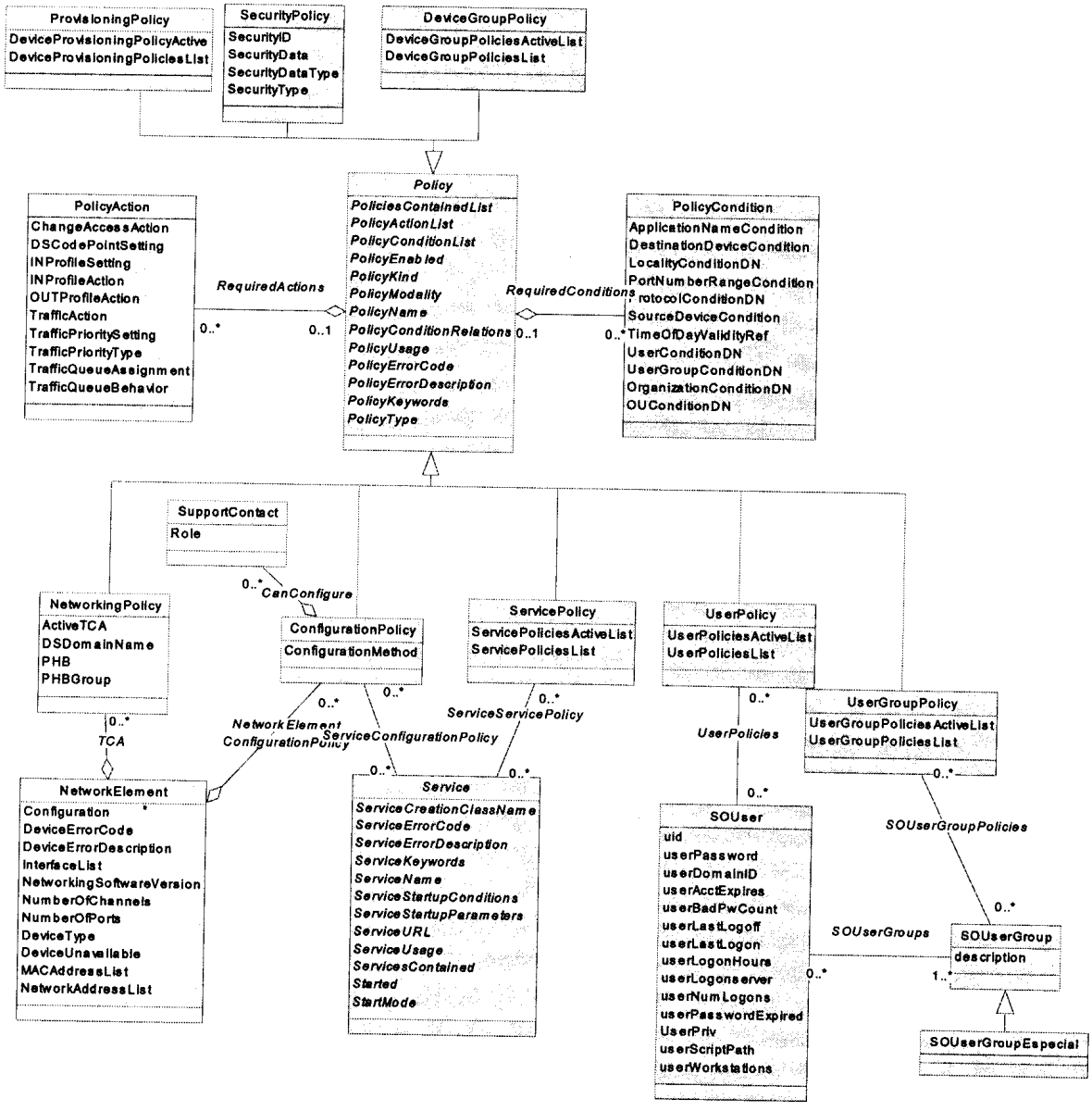

Figura 5.18 – Diagrama de Classes de Políticas de Segurança para o Ambiente da Cimento Rio Branco S/A

Com a aplicação destes diagramas de classes, o esquema para dar suporte aos serviços/aplicações de segurança estarão preparados.

6. DISTRIBUIÇÃO/REPLICAÇÃO DO ESQUEMA DO DIRETÓRIO

O esquema do diretório pode ser distribuído ou centralizado dependendo da necessidade de cada ambiente. Em ambientes complexos com vários sites espalhados em locais físicos distantes, a utilização da distribuição ou replicação se torna necessária. Por distribuição do esquema do diretório entende-se dividir a árvore do diretório em sub árvores e alocá-las em locais físicos diferentes. Por replicação entende-se copiar sub árvores ou toda a árvore para outros locais físicos mantendo esta cópia sincronizada com a cópia original.

Para a aplicação da distribuição, é necessário que se faça um estudo aprofundado em todo o ambiente para verificar as necessidades particulares em cada local onde as informações do diretório serão necessárias. Desta forma, poderá ser definido como a árvore do diretório deverá ser subdividida para atender as necessidades identificadas. Definida a distribuição do esquema, devem ser implementados mecanismos que permitam a realização de pesquisas de dados em servidores distintos para que, desta forma, requisições de dados que não possam ser atendidas por um determinado servidor possam ser repassadas para o servidor responsável pelas informações. Para verificar mais detalhes sobre estes mecanismos veja [NETDEP99].

Feita a distribuição e implementados, os mecanismos de comunicação/pesquisa (*referrals*) entre os servidores, os administradores do ambiente, com o passar do tempo, podem verificar a necessidade da implementação da replicação de partes do esquema. Isto é identificado quando existe muita consulta sendo realizada entre os sites do ambiente. Esta grande quantidade de consulta, leva a conclusão de que os dados do site que está sendo muito consultado devem ser replicados para os sites que realizam as consultas.

Como descrito anteriormente, a replicação é o mecanismo que permite que os dados do diretório de um servidor sejam copiados automaticamente para outro servidor. Utilizando a replicação, pode-se melhorar o desempenho de acesso às informações armazenadas no diretório, pois é possível colocar os dados mais próximos dos usuários. Um outro benefício da replicação é que ela permite

aumentar a disponibilidade, pois replicando a árvore do diretório (esquema e dados), ou parte dela, pode-se garantir que mesmo ocorrendo uma falha de hardware, software ou rede em uma instância do diretório outras instâncias estarão disponíveis.

A arquitetura de replicação utiliza servidores com papéis distintos. Existem os servidores que possuem a cópia principal de um objeto, chamados por este trabalho de fornecedores, e servidores que recebem uma cópia deste objeto, chamados por este trabalho de consumidores. É importante notar que um determinado servidor pode ser fornecedor de alguns objetos e ao mesmo tempo consumidor de outros objetos, permitindo que a árvore global do diretório seja distribuída de acordo com as necessidades de implementações específicas.

Com relação ao gerenciamento das alterações nos dados do diretório podem existir dois tipos de implementação: mestre-escravo e mestre-mestre. Na implementação mestre-escravo o servidor de diretório fornecedor é responsável por realizar toda e qualquer alteração nos dados, ou seja, mesmo que uma alteração, adição, eliminação de um dado do diretório seja requisitada para um servidor consumidor esta requisição deve referenciar o servidor fornecedor pois a replicação sempre será no sentido fornecedor para consumidor. Já na implementação mestre-mestre as alterações podem ser realizadas tanto no servidor de diretório fornecedor como no consumidor pois estas alterações poderão ser replicadas tanto no sentido fornecedor para consumidor como no sentido consumidor para fornecedor.

A escolha do tipo de implementação vai depender da natureza do ambiente onde a replicação vai ser executada. Em ambientes onde existe a necessidade de uma maior flexibilidade a implementação mestre-mestre é a mais indicada, pois mesmo que o servidor fornecedor esteja indisponível por algum motivo (Ex.: falha na rede) será possível que alterações sejam realizada nos dados do diretório. Já em ambientes onde se deseje um menor grau de complexidade de administração dos dados a implementação mestre-escravo é mais indicada, pois preocupações com conflitos na consistência dos dados devido a alterações poderem ser realizadas em várias cópias não existe.

Outro ponto importante que deve ser analisado na replicação é o sincronismo. O sincronismo ou sincronização é o ato de iniciar a cópia/atualização das informações do diretório. Existem duas formas de se implementar o sincronismo: replicação iniciada pelo servidor fornecedor e replicação iniciada pelo servidor consumidor.

Replicação do Esquema

Na replicação do esquema do diretório podem ser utilizadas várias configurações. A configuração mais básica é onde o servidor fornecedor replica toda a árvore do diretório para um ou mais servidores consumidores utilizando a arquitetura mestre-escravo ou mestre-mestre. A quantidade de servidores consumidores vai depender da velocidade dos links de rede entre os servidores consumidores e o servidor fornecedor e a quantidade de dados a ser replicada. Um exemplo é demonstrado na figura abaixo:

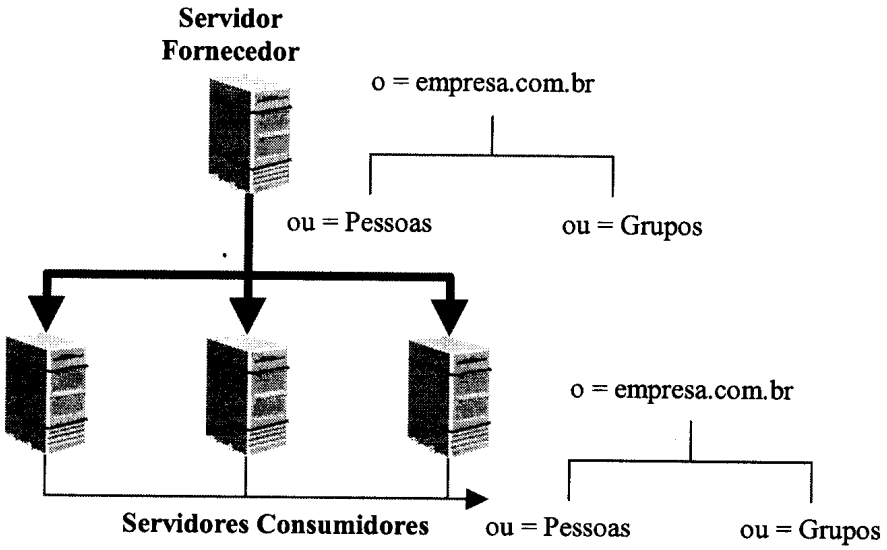

Figura 6.1 - Replicação Básica

Um recurso que pode ser utilizado na replicação de dados é o cascateamento da replicação. Este recurso permite que se replique os dados de um servidor fornecedor para um consumidor e depois este consumidor replica os dados para outro consumidor. É adequado utilizar este recurso em ambientes

onde entre os servidores existam links de comunicação melhores do que outros. Um exemplo é apresentado na figura 6.2:

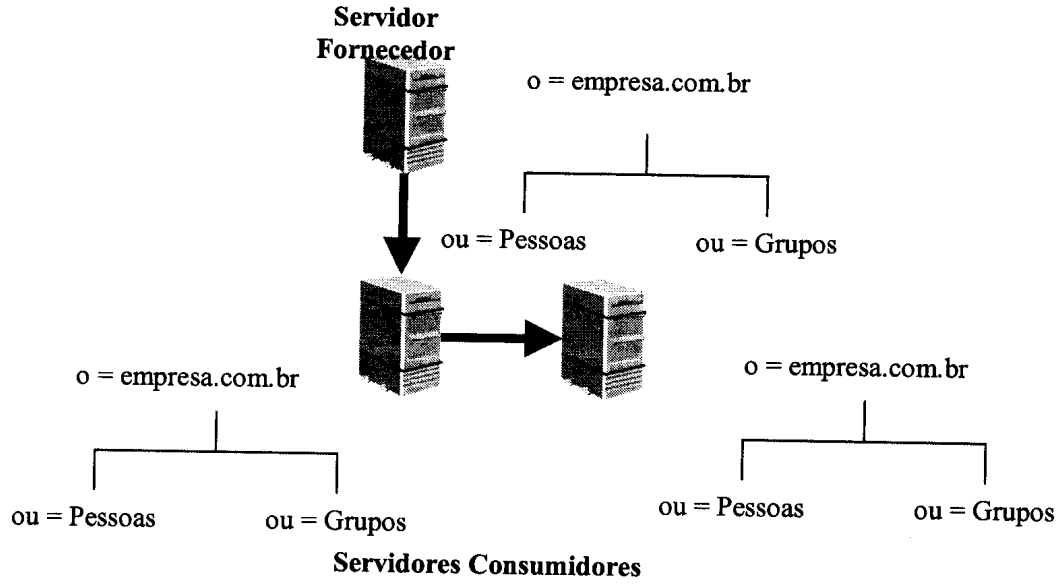

Figura 6.2 - Cascateamento da Replicação

A figura 6.2 mostra que um servidor fornecedor replica as informações para um servidor consumidor e este replica para outro servidor consumidor.

Um servidor fornecedor pode replicar apenas uma parte da árvore do diretório. Um exemplo, é um ambiente onde exista a matriz da empresa, onde fica a árvore global, e filiais onde existe apenas os dados referentes a filial. Desta forma o administrador do diretório global pode delegar poderes administrativos sobre estas sub árvores para os administradores locais de cada filial, otimizando assim a administração dos dados do diretório. Por exemplo, pode-se replicar os dados referentes aos usuários e recursos compartilhados da rede específicos para cada filial, permitindo que a administração de segurança destes objetos seja realizada pelos administradores locais. Esta situação pode ser verificada na figura 6.3 a seguir:

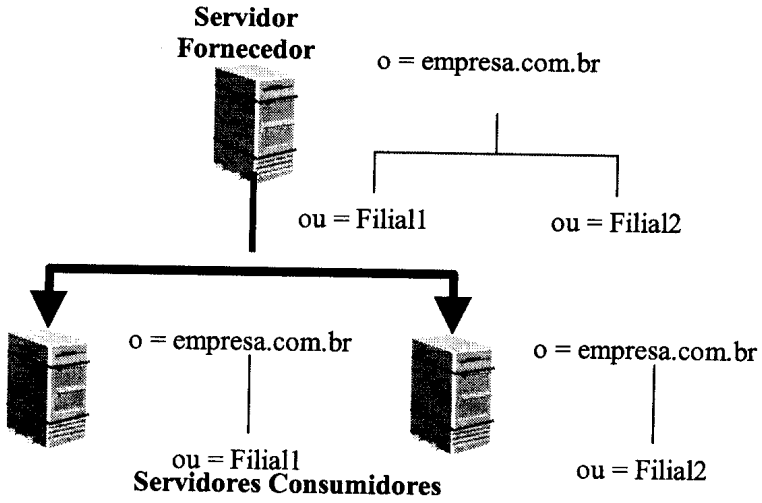

Figura 6.3 - Replicação Parcial da Árvore de Diretório

Outro ponto importante na replicação é a segurança dos dados no momento da replicação. Isto é ainda mais crítico caso os dados desta replicação passem por uma rede pública como a Internet. Para resolver este problema, o serviço de diretório deve possuir mecanismos de criptografia da informação. Caso não possua, deve ser analisada a utilização de uma Virtual Private Network (VPN) para que os dados da replicação sejam protegidos. Para maiores informações sobre VPNs veja [GON97].

Depois de analisadas todas estas considerações sobre distribuição/replicação, (mais informações podem ser buscadas em [NETDEP99]) pode ser definido como as informações de segurança do esquema analisado por este trabalho devem ser tratadas. Basicamente estas informações devem ser replicadas para todos os sites de um mesmo domínio administrativo. A utilização da replicação é necessária pois estas informações devem estar situadas em todos os locais que elas sejam necessárias.

Por exemplo, considere um site a.com e um site b.com de um mesmo domínio administrativo situados em locais distintos com seus respectivos servidores de diretório. Veja a figura a seguir:

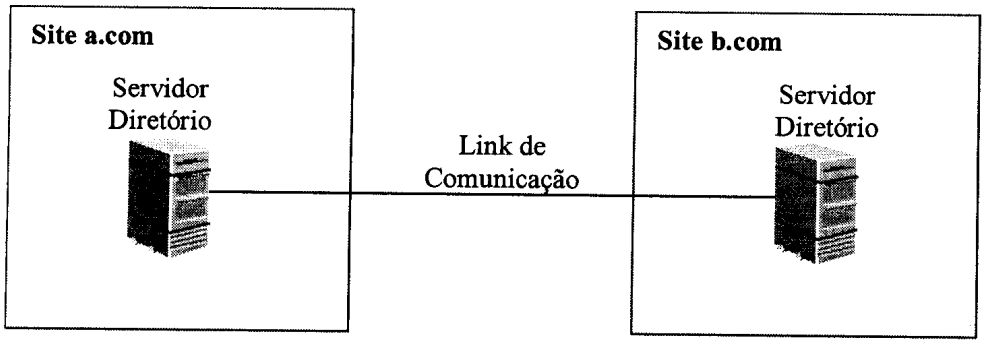

Figura 6.4 - Replicação dos Dados de Segurança

Suponha que as informações de segurança estejam centralizadas no servidor de diretório do site a.com e no momento que um usuário do site b.com vai ser autenticado o link de comunicação entre os site tem algum problema. Neste caso, o usuário ficar impossibilitado de trabalhar pois não consegue se conectar a rede. Isto vale para autenticação, autorização e política de segurança.

É importante que todas as informações referentes a autenticação, autorização e políticas estejam situados em ambos os locais. O que pode ser feito, e isto depende das características do ambiente, é replicar a estrutura da árvore do diretório e distribuir apenas os seus dados. Ou seja, tanto o site a.com como o site b.com terão a mesma estrutura mas o site a.com terá apenas os dados de autenticação, autorização e políticas referentes a seus recursos/usuários o mesmo acontecendo com o site b.com. Isto poderá gerar problemas se os usuários dos sites a.com e b.com se deslocarem constantemente entre estes locais. Por exemplo, caso um usuário do site a.com vá para o site b.com e o link de comunicação não estiver funcionando, este não poderá se autenticar.

7. CONCLUSÃO

O esquema apresentado nesse trabalho, composto dos diagramas de classe para autenticação, autorização e políticas de segurança, pode ser implementado em qualquer ambiente que suporte um serviço de diretório. A utilização de um repositório global de informações traz benefícios claros em relação a administração da segurança do ambiente computacional, pois permite estabelecer relacionamentos entre as aplicações, serviços, dispositivos de redes, políticas de segurança e usuários.

A estruturação das informações dentro do serviço de diretório é altamente escalonável, pois a "quebra" da árvore do diretório é bem flexível permitindo que se execute a fragmentação da informação de forma simples. Outro ponto importante é a eficiência das consultas LDAP, que permitem a busca de informações no diretório de forma simples e eficaz.

Apesar da aplicação de um esquema global ser benéfica do ponto de vista da consistência das informações, um grande esforço resta a ser feito no sentido de adaptar as aplicações e serviços de gerenciamento existentes ao conceito de diretório.

A utilização da gerência de redes baseada no serviço de diretório necessita que o administrador modele sua rede a partir de um "esquema base" pré-definido, como o proposto no capítulo 5 desse trabalho. Essa etapa de modelagem constitui um obstáculo potencial a utilização do serviço de diretório, pois os administradores atuais precisarão de um treinamento especializado para trabalhar com essa forma de representação. Provavelmente, as empresas precisarão contar com profissionais especializados voltados exclusivamente na manutenção das informações de seu serviço de diretório.

Deve-se considerar também os aspectos relacionados à eficiência do serviço de diretório. Em geral, os diretórios são destinados a funcionar como repositórios de informações não voláteis, isto é, informações que sofrem muita leitura mas pouca alteração. Contudo, os elementos de rede nem sempre se encaixam nesta categoria. Por exemplo, a informação relacionada ao tráfego que atravessa um dado dispositivo de rede é uma informação extremamente volátil. Desta forma, devido a questões de desempenho, pode ser que exista a necessidade de se manter muitos atributos de estado fora do diretório.

8. BIBLIOGRAFIA

[ADS97] MICROSOFT. Active Directory Technical Summary. Microsoft White Paper, 1997.

[ANA97] Ananthanpillai, Raj. Implementing Global Networked Systems Management: Strategies and Solutions. McGraw-Hill, 1997

[ZVER00] Zvericky, E. D; Cooper, S. ; Chapman, D. B. Building Internet Firewalls, 2nd Edition. O'Reilly & Associates, 2000.

[BRAIN96] Brain, Marshall. Win32 System Services – The Heart of Windows & Windows NT. Prentice Hall, 1996.

[CIM99] Distributed Management Task Force, Inc. *Common Information Model* (CIM) Specification Version 2.2, June 1999.
URL: http://www.dmtf.org/spec/cim_spec_v22/

[DEN99] Distributed Management Task Force, Inc. *Directory-enabled Networks* (DEN) - Information Model and Base Schema Specification Version 3.0c5, 1999
URL: http://www.dmtf.org/

[FER98] Ferguson, Paul. Quality of Service - Delivering QoS on the Internet and in Corporate Networks. Wiley Computer Publishing, 1998

[FOW97] Fowler, Martin. UML Distiled - Applying the Standard Object Modeling Language. Addison-Wesley, 1997

[GLE97] GLEN, Bruce. Security in Distributed Computing. Did You Lock *the Door ?* Prentice Hall, 1997.

[GON97] GONÇALVES, Marcus. *Firewalls Complete*. MC Graw Hill, 1997.

[HAR96] HARNEDY, Sean. *Total SNMP*. Prentice-Hall, 1997.

[HOWE97] T. Howes, M. Smith, LDAP: Programming Directory-Enabled Applications with LightweightDirectory Access Protocol. Macmillan Technical Publishing, 1997.

[ITUX500] ITU-T, Recommendations X.500, The Directory - Overview of Concepts, Models, and Services, ITU-T Blue Book, 1988.

[NDS98] NOVELL. NDS Personality - NDS White Paper, Novell Inc, 1998. URL: http://novell.com/products/nds/wpnds.html

[NET96] NETSCAPE Corporation. The SSL Protocol Version 3.0. Netscape Inc, 1996.

[NET97] NETSCAPE Corporation. An Internet Approach to Directories. Netscape Inc, 1997.
URL: http://developer.netscape.com/docs/manuals/ldap/index.html.

[NETDEP99] NETSCAPE Corporation, Netscape Directory Server Deployment Guide, Netscape Inc, 1999.

[RFC1155] McCLOGHRIE, Keith & ROSE, Marshall, "*Structure and* Identification of Management Information for TCP/IP-based Internets", Request for Comments 1155, FTP=ds2.internic.net,1990.

[RFC1157] CASE, Jeffrey et all, "A Simple Network Management Protocol *SNMP*", Request for Comments 1157, FTP=ds2.internic.net, 1990.

[RFC1212] ROSE, Marshall & McCLOGHRIE, Keith, "*Concise MIB Definitions*", Request for Comments 1212, FTP=ds2.internic.net, 1991.

[RFC1215] ROSE, Marshall, "A Convention for Defining Traps for Use with *the SNMP*", Request for Comments 1215, FTP=ds2.internic.net, 1991.

[RFC1510] J. Kohl, C. Newman. The Kerberos Authentication Service (v5). Request For Comments 1510, Setember 1993.
URL: ftp://ds.internic.net/rfc/rfc1510.txt

[RFC1777] W. Yeong, T. Howes, S. Kille, *Lightweight Directory Access Protocol.*Request For Comments 1777, March 1995.
URL: ftp://ds.internic.net/rfc/rfc1777.txt

[RFC2251] Wahl, M., T. Howes, and S. Kille, *Lightweight Directory Access Protocol* (v3) Request For Comments 2251, December 1997.
URL: ftp://ds.internic.net/rfc/rfc2251.txt.

[SEM98] C. Semeria, F. Fuller. Policy-Powered Networking and the Role of *Directories.* 3COM White Paper 1998.
URL: http://www.3com.com/technology/tech_net/white_papers/500665b.html

[STA96] STALLINGS, William. SNMP SNMPv2 and RMON Pratical *Network Management.* Massachussets: Addison Wesley, 1996.

[TAN96] TANENBAUM, Andrew S. *Computer Networks* Third Edition Prentice-Hall, 1996.

ANEXO 1 – VISÃO GERAL DE CLASSES E RELACIONAMENTOS DE UM ESQUEMA DE DIRETÓRIO QUE SUPORTE INFORMAÇÕES DA REDE

Este anexo demonstra a hierarquia de algumas das classes CIM, essenciais para o DEN, e a hierarquia das classes DEN. A hierarquia das classes X.500 pode ser verificada na especificação X.500 de 1993.

Esta análise visa verificar apenas as principais classes e relacionamentos da especificação através de um modelo hierárquico e quando apropriado um modelo de objetos. Para descrições mais detalhadas das classes, atributos, relacionamentos e diagramas, analise as especificações CIM e DEN.

Hierarquia de Classes CIM

Abaixo segue uma descrição de cada hierarquia dentro do esquema DEN proposto pelo DMTF.

8.1.1. Hierarquia da Classe ManagedSystemElement

A classe CIM ManagedSystemElement é a classe base para as classes PhysicalElement e LogicalElement. Ela fornece a noção fundamental de administração de um sistema ou componente de um sistema.

8.1.2. Hierarquia da Classe PhysicalElement

A classe PhysicalElement é a classe origem (root) para definição de todos os componentes do sistema que possuam uma identidade física.

Esta hierarquia de classes é representada abaixo:

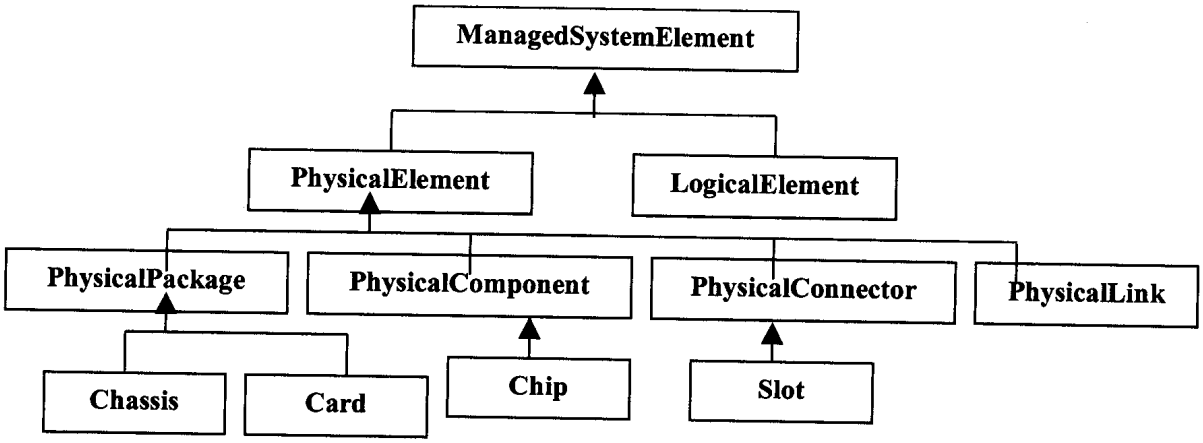

Figura a1.1 - Hierarquia da Classe PhysicalElement

Diagrama de Classe de Elementos Físicos

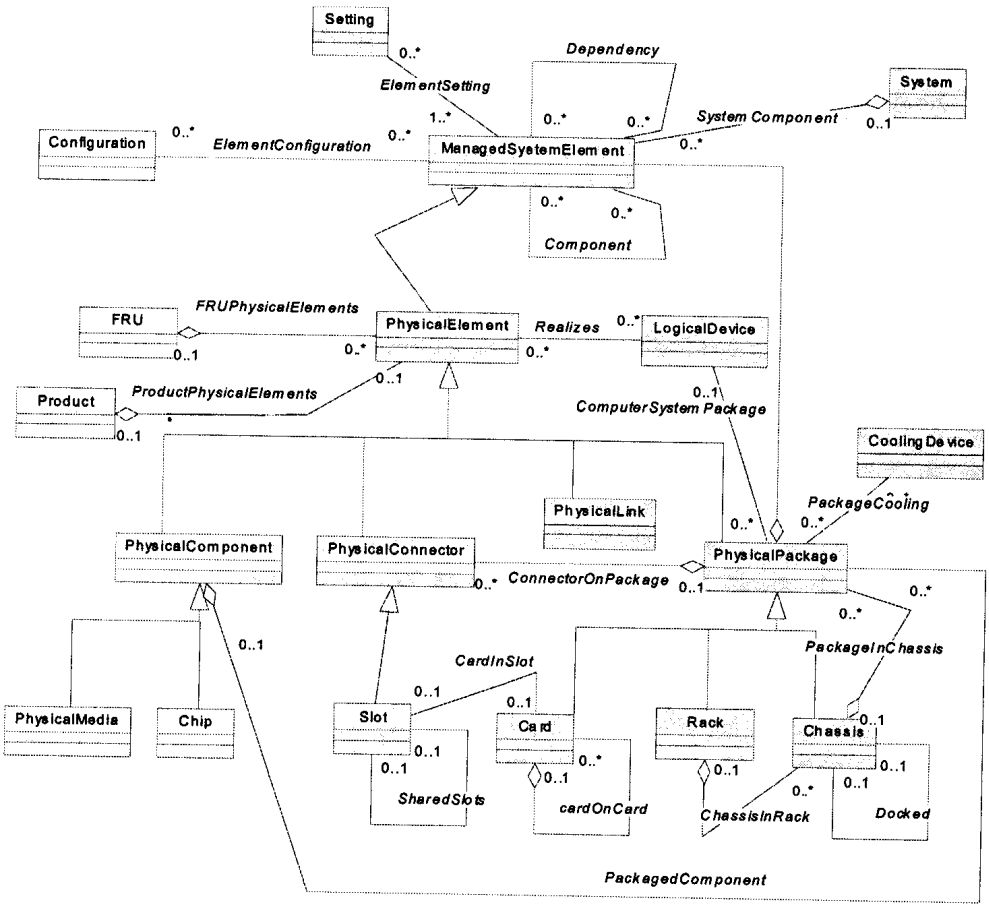

Figura a1.2 - Diagrama de Objetos para Elementos Físicos

8.1.3. Hierarquia da Classe LogicalElement

A classe LogicalElement é a origem para se definir qualquer componente do sistema que realiza uma ou mais funções lógicas.

Esta hierarquia de classes é representada abaixo:

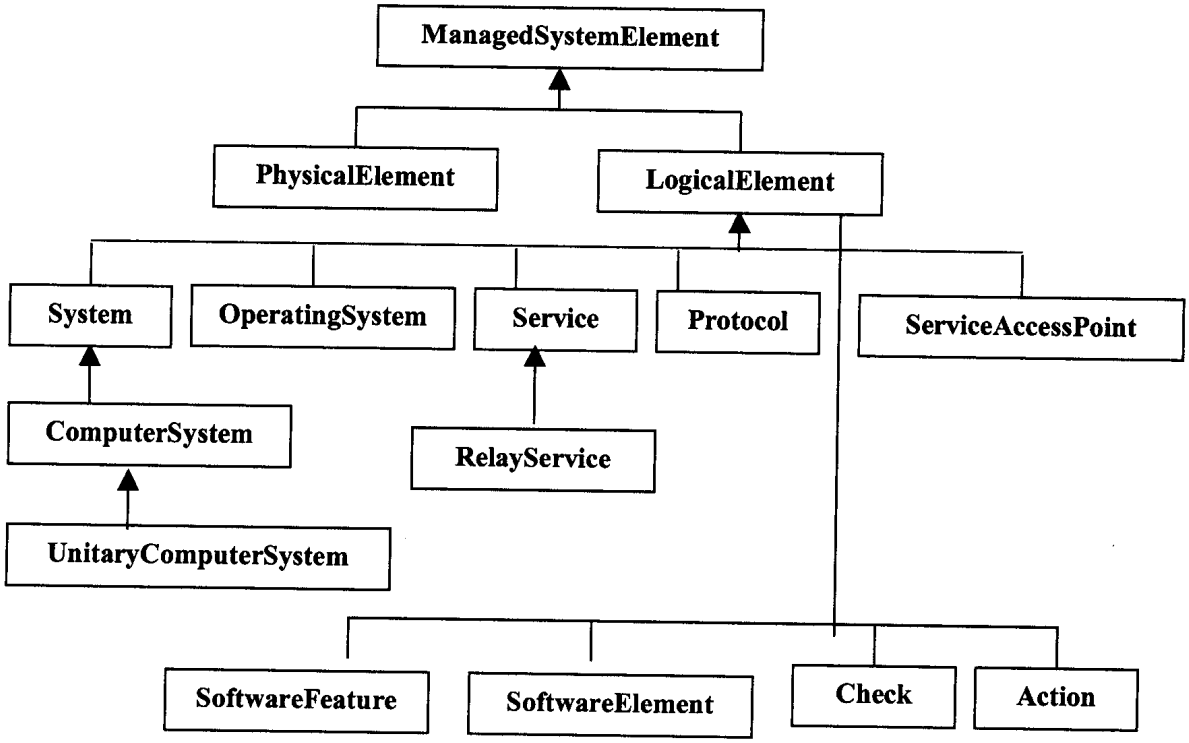

Figura a1.3 - Hierarquia da Classe LogicalElement

Diagrama de Classes de Elementos Lógicos

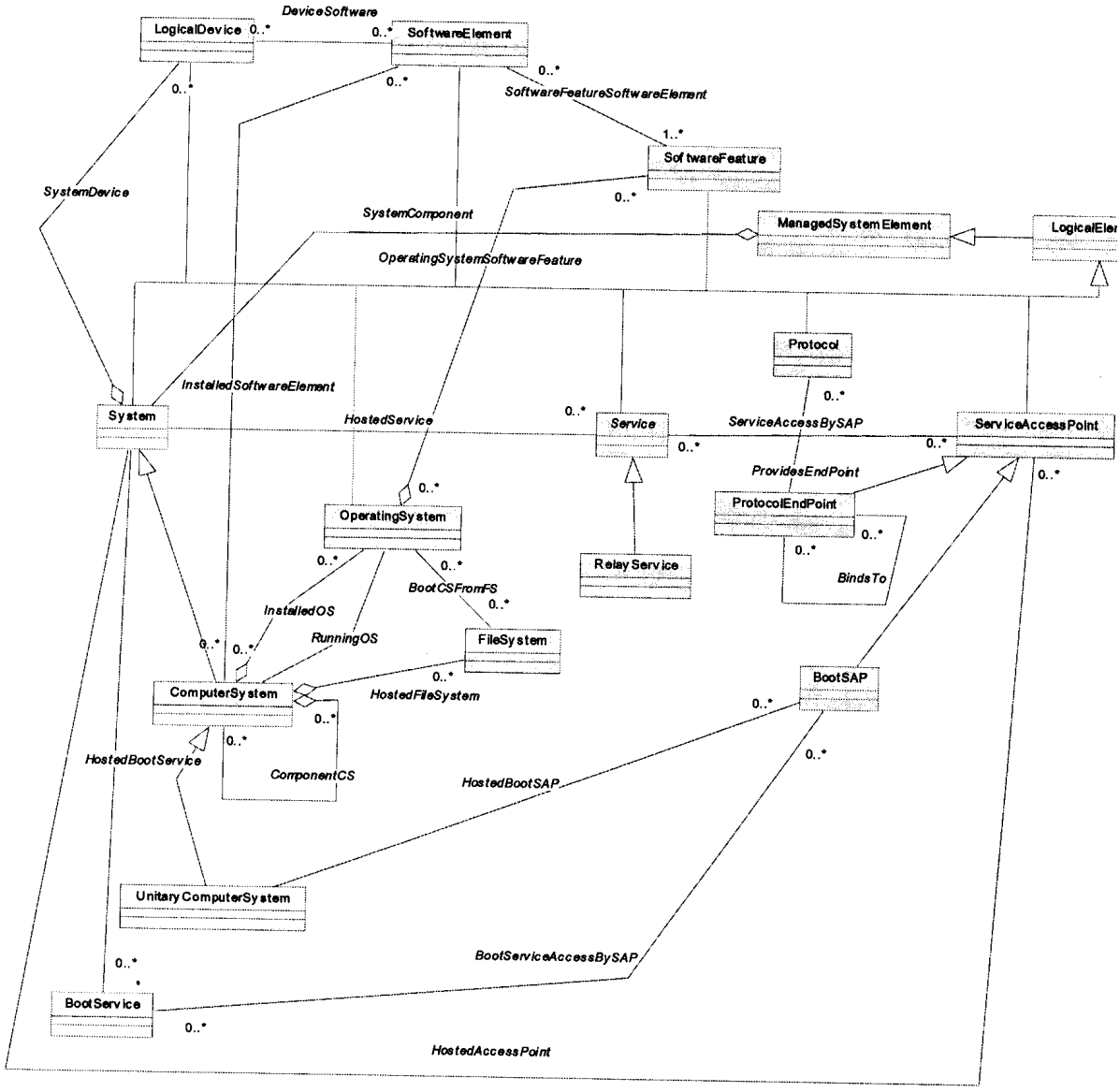

Figura a1.4 - Diagrama de Objetos de Elementos Lógicos

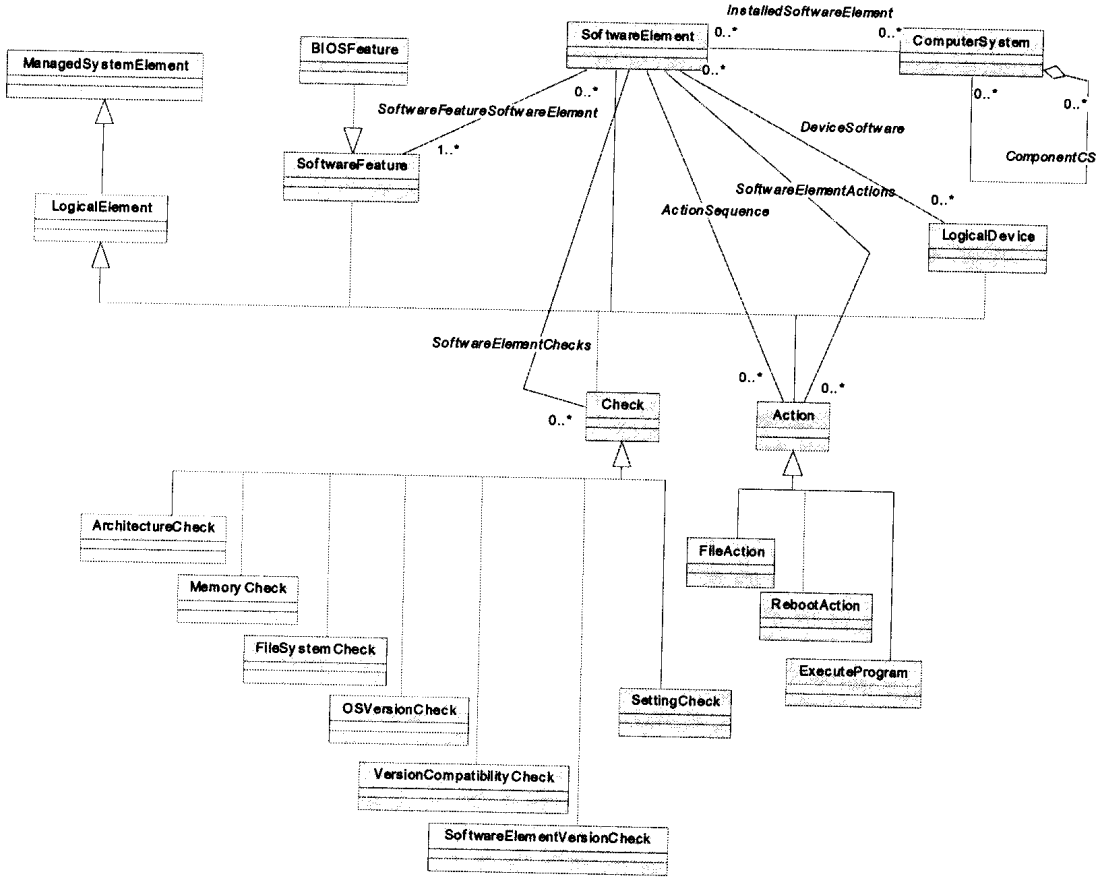

Figura a1.5 - Diagrama de Objetos de Elementos Lógicos (cont.)

Hierarquia de Classes DEN

Agora será mostrado as classes que foram definidas pela especificação DEN.

8.1.4. Hierarquia da Classe Service

Esta classe pode ser utilizada (através de especialização) para definir objetos serviço de diferentes tipos que estão disponíveis na rede. Esta hierarquia está sujeita a alterações pelas revisões da especificação DEN.

Esta hierarquia de classes é demonstrada abaixo:

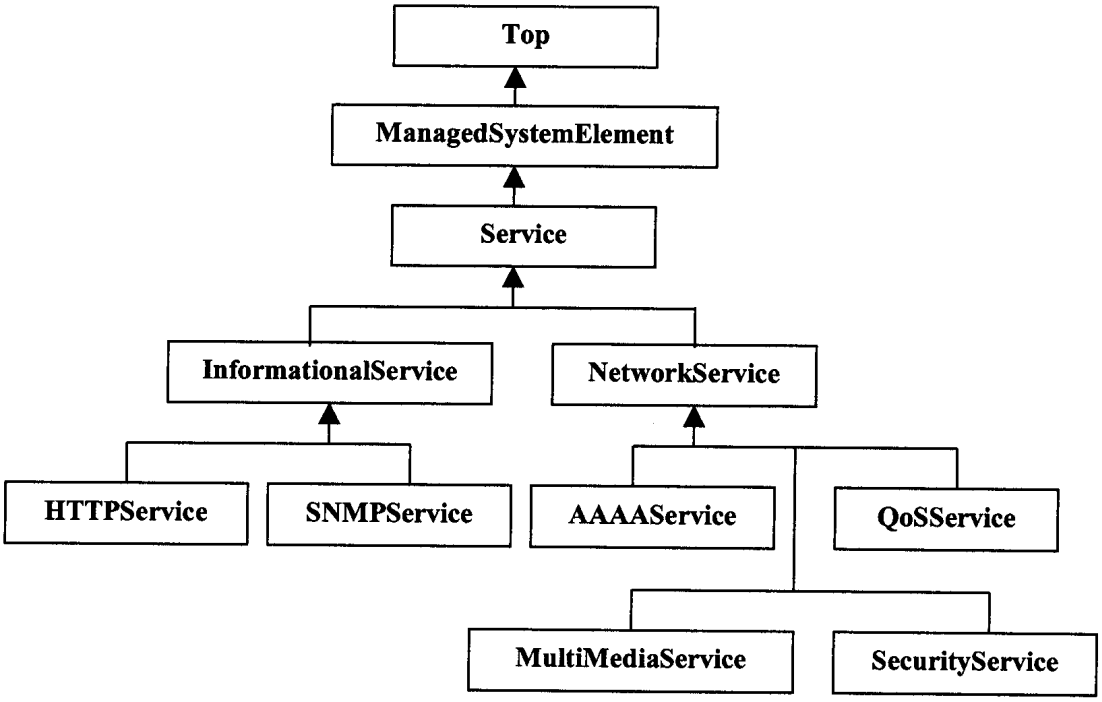

Figura a1.6 - Hierarquia da Classe Service

Diagrama de Classes de Serviços

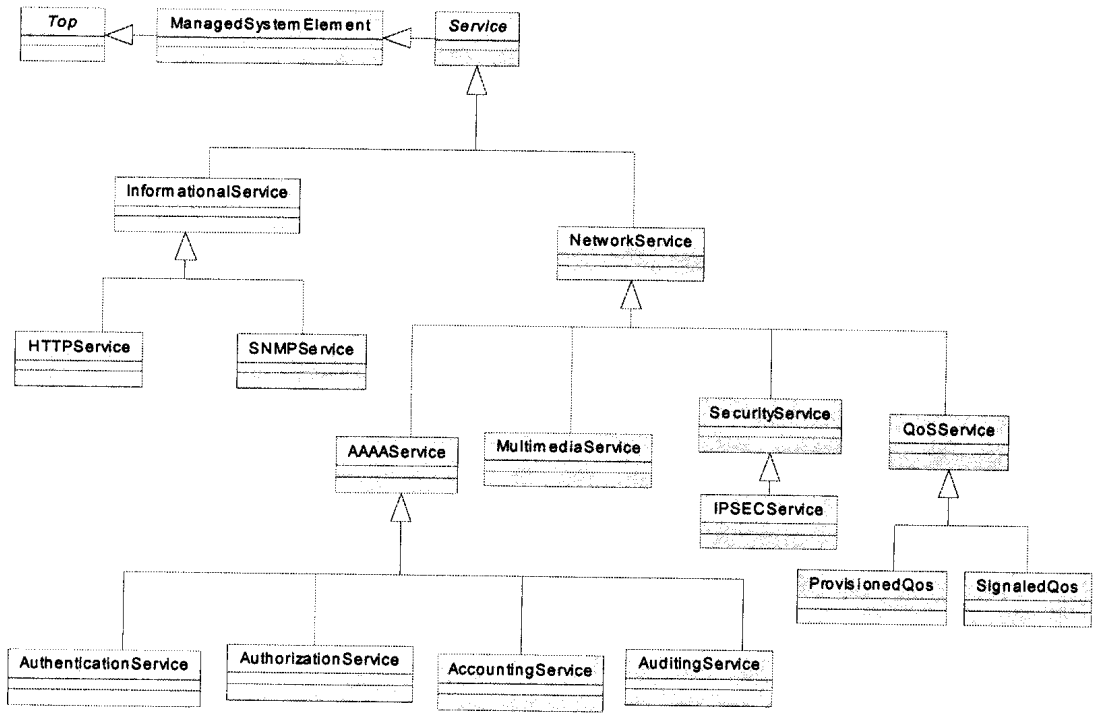

Figura a1.7 - Diagrama de Classes de Serviços

8.1.5. Hierarquia da Classe Profile

Esta hierarquia é expansível para definição de objetos profile que permitem representar Profiles de diferentes tipos. Profiles definem características e necessidades de um objeto.

A hierarquia é demonstrada abaixo:

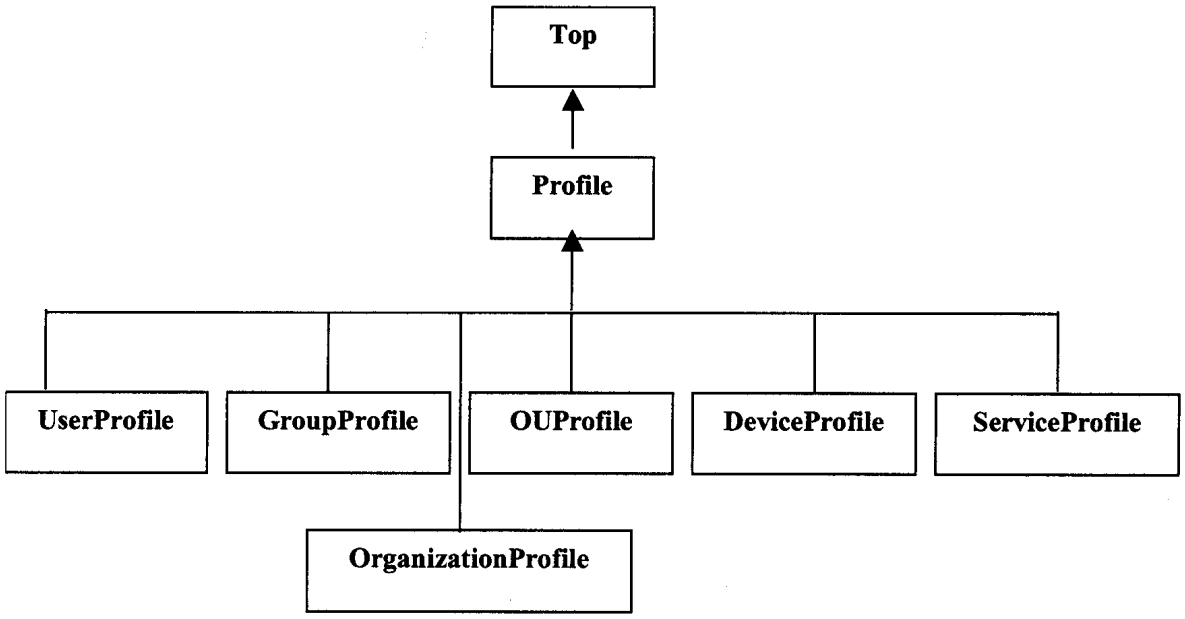

Figura a1.8 - Hierarquia da Classe Profile

Diagrama de Classes de Profile

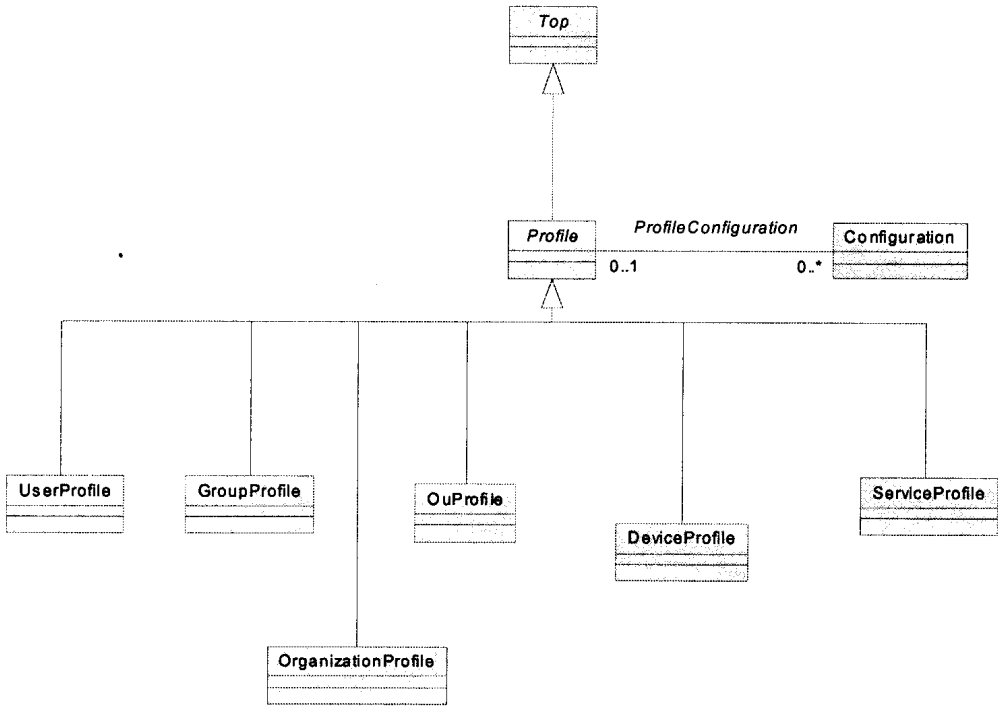

Figura a1.9 - Diagrama e Classes de Profiles

8.1.6. Hierarquia de Classes Policy

Esta hierarquia de classes é utilizada para definir objetos de políticas que permitem a desenvolvedores representar políticas de diferente tipos.

As políticas consistem em dois componentes:

Conjunto de condições nas quais a aplicação da política deve ser especificada (contexto da política)

Conjunto de ações que deve manter o estado corrente do objeto ou mudá-lo para um novo estado por conseqüência da execução do conjunto de condições.

As políticas podem ser classificadas em duas categorias:

Políticas de Serviço (Service Policies): Descrevem serviços disponíveis na rede.

Políticas de Uso (Usage Policies): Descrevem quais políticas utilizam quais serviços quando as políticas são satisfeitas. Políticas de uso descrevem mecanismos particulares empregados para manter o estado corrente do objeto ou para mover o objeto para um novo estado para que este utilize os serviços especificados.

Para se entender a hierarquia de classes, é necessário entender o contexto no qual a hierarquía é utilizada. Existem quatro partes importantes na arquitetura de políticas baseadas na especificação DEN:

Administração, feita através de um ponto de administração logicamente centralizado.

Repositório. Servidor de diretório compatível com DEN.

Pontos de decisão de políticas. Existe pelo menos um em cada domínio administrativo.

Pontos de execução de políticas. Existem vários por domínio administrativo. Cada conjunto de pontos de execução de políticas é controlado por um ponto de decisão de políticas.

A figura 5.10 abaixo demonstra que as políticas são administradas através de um ponto de administração logicamente centralizado que baseia-se em um diretório DEN para acesso a dados comuns como profile de usuários. Os servidores de políticas (ex.: pontos de decisão de políticas) também utilizam o diretório DEN para buscar as porções estáticas das políticas. Os dispositivos (pontos de execução) se comunicam com os servidores de políticas, mas não necessariamente com o diretório DEN.

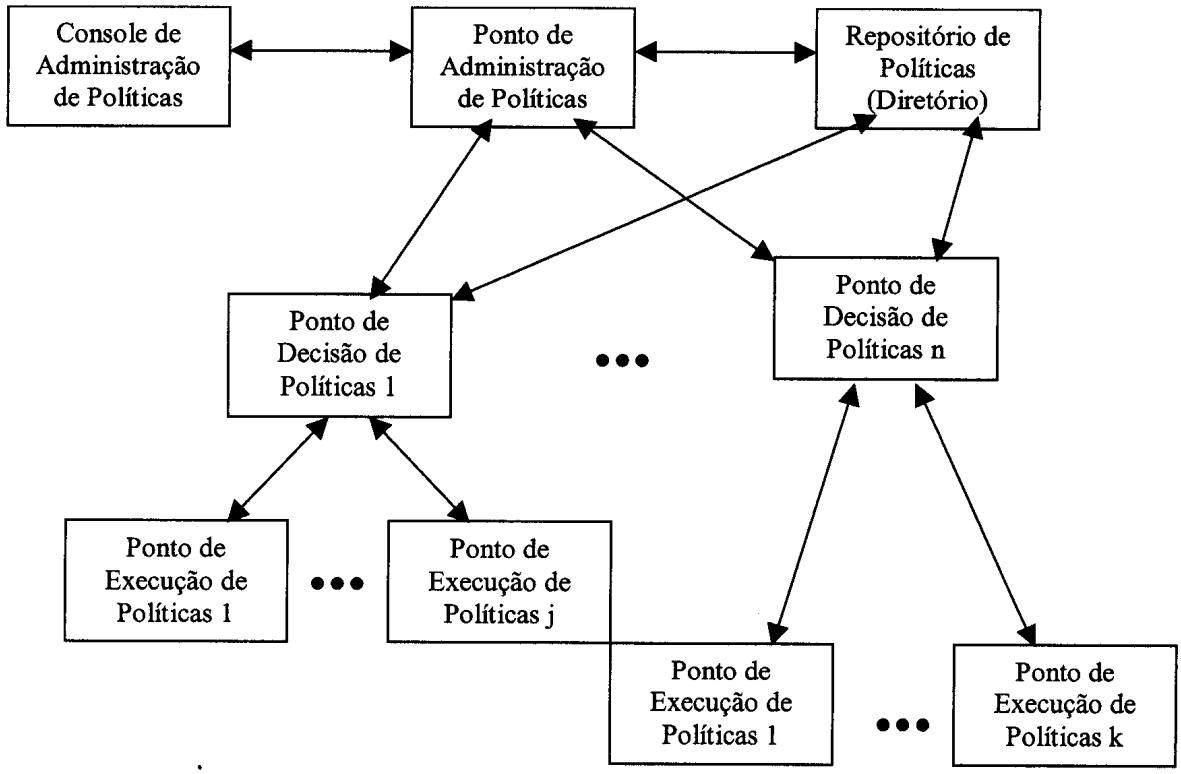

Figura a1.10 - Componentes para Execução de Políticas

Políticas permitem que se controle e administre serviços de rede. Políticas DEN são conjuntos de regras gerais que governam as operações da rede e desenvolvimento de serviços. Além disso, a especificação DEN fornece classes que habilitam a navegação em aplicações específicas entre diferentes domínios de conhecimento que refletem uma visão única da política. Por exemplo, a especificação de um política requer que o administrador altere a ACL de um dispositivo. Isto requer a especificação de um administrador de rede que possua a autoridade de alterar ACLs naquele dispositivo. Isto é refletido no modelo lógico. Já que o dispositivo também está especificado neste modelo, ele pode ser

utilizado para transferir para o modelo físico para determinar onde o dispositivo está localizado fisicamente.

As políticas possuem atributos estáticos e dinâmicos. Os atributos estáticos representam os direitos básicos e privilégios dos objetos. Isto pode ser baseado no papel do objeto, ou parâmetros como localização. Em contrapartida, atributos dinâmicos são de interesse na execução da política (se alteram constantemente).

A hierarquia de classes é representada abaixo:

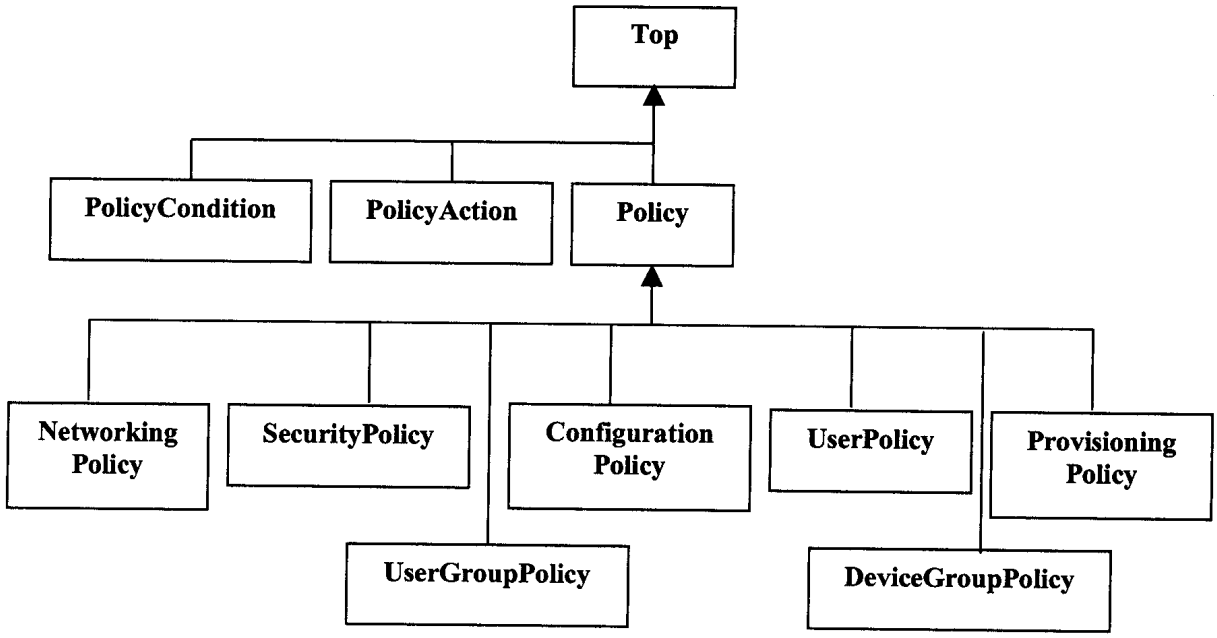

Figura a1.11 - Hierarquia da Classe Policy

Diagrama de Classe de Políticas

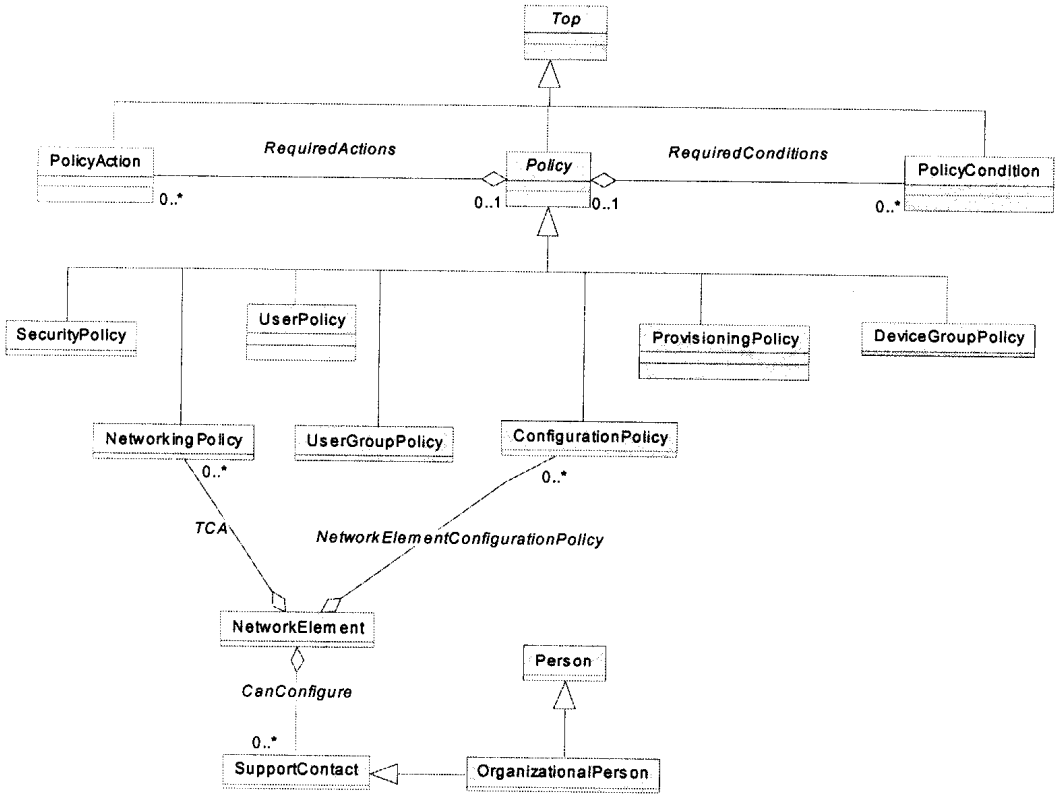

Figura a1.12 - Diagrama de Classes para Políticas

8.1.7. Hierarquia da Classe Device

O esquema .de dispositivos (device) DEN mantém compatibilidade com o X.500 além de incorporar vários conceitos da versão 2.0 do CIM (Common Information Model) do DMTF. O X.500 define a classe Device como a super classe de todos os dispositivos físicos, mas não adiciona nenhuma estrutura a esta definição. O esquema CIM refina esta definição em duas subclasses chamadas PhysicalElement, que representa todos os aspectos físicos de um dispositivo e LogicalElement, que representa as capacidades funcionais deste dispositivo. O CIM especifica um rico esquema de classes para representar tanto aspectos físicos como lógicos dos dispositivos. Uma visão geral destas classes foi apresentado anteriormente por este trabalho. Para um nível maior de detalhamento consulte a especificação CIM.

A especificação DEN segue este modelo representando um elemento de rede como tendo características físicas e lógicas. A hierarquia de classes recomendada para representar as características físicas e lógicas dos elementos de rede são apresentadas a seguir.

8.1.8. Hierarquia da Classe NetworkPackage

A especificação CIM não considera os requisitos de dispositivos e serviços de rede em detalhes, desta forma, peca nas funcionalidades necessárias para se modelar dispositivos e serviços de rede. Para suprir estas dificuldades, há a necessidade de se adicionar algumas funcionalidades na forma de atributos, relacionamentos e métodos em duas áreas: conceitos físicos de compartimentos implementados em CIM pela classe PhysicalPackage e adições a funcionalidade da classe Card que é encaixada dentro de um container. Para cada uma destas áreas a especificação DEN adota as seguintes soluções:

Para containers físicos a especificação DEN adiciona uma subclasse a PhysicalPackage chamada NetworkPackage para fornecer as funcionalidades necessárias.

Para placas. de rede a especificação DEN adiciona alguns atributos e relacionamentos às classes da especificação CIM.

As alterações necessárias em PhysicalPackage estão nas seguintes áreas:

Cabling: requisitos de cabeamento dos dispositivos de rede, especialmente aqueles onde um chassi pode se conectar em outro chassi, são mais complexos do que os previstos pela especificação CIM.

Força e Cooling: a especificação CIM prevê estes conceitos, mas estes precisam ser mais flexíveis.

Backplanes: vários dispositivos de rede possuem backplanes de alta velocidade que fornecem funcionalidades especiais com o dispositivo. Isto precisa ser modelado.

FRUs Especiais: algumas FRUs são montadas diretamente no chassis. A especificação CIM está limitada em montar componentes em uma placa e conectar a placa (Card) a um Slot que faz parte de um Chassis. Neste caso existe uma placa mas nenhum Slot e esta capacidade precisa ser modelada.

Em relação a classe Card algumas flexibilidades em relação a configuração são necessárias. As placas de rede são de dois tipos: configuráveis e não configuráveis. Várias opções de configuração não são previstas pelo CIM e para solucionar este problema um conjunto adicional de atributos e relacionamentos devem ser adicionados a certas classes da especificação CIM.

A hierarquia resultante destas alterações é demonstrada na figura 5.13 abaixo:

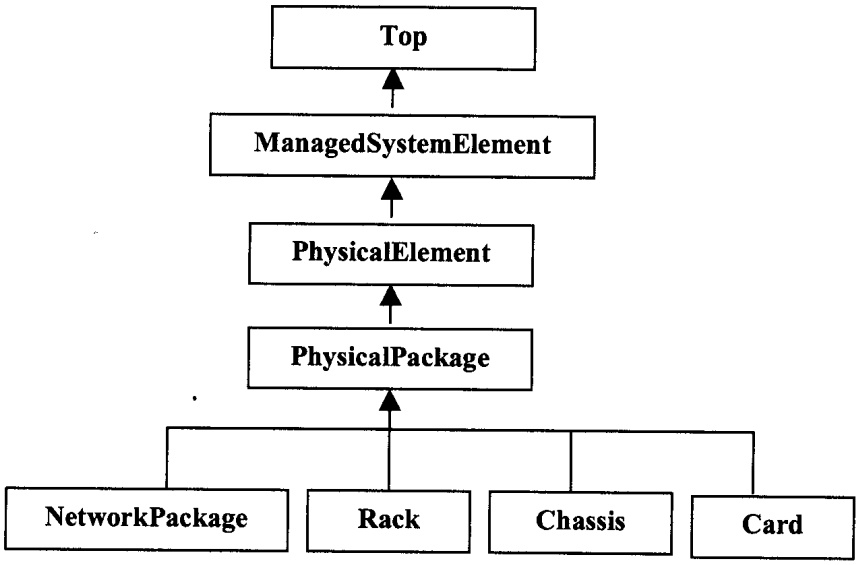

Figura a1.13 - Hierarquia da Classe NetworkPackage

Diagrama de Classe para Pacotes de Rede

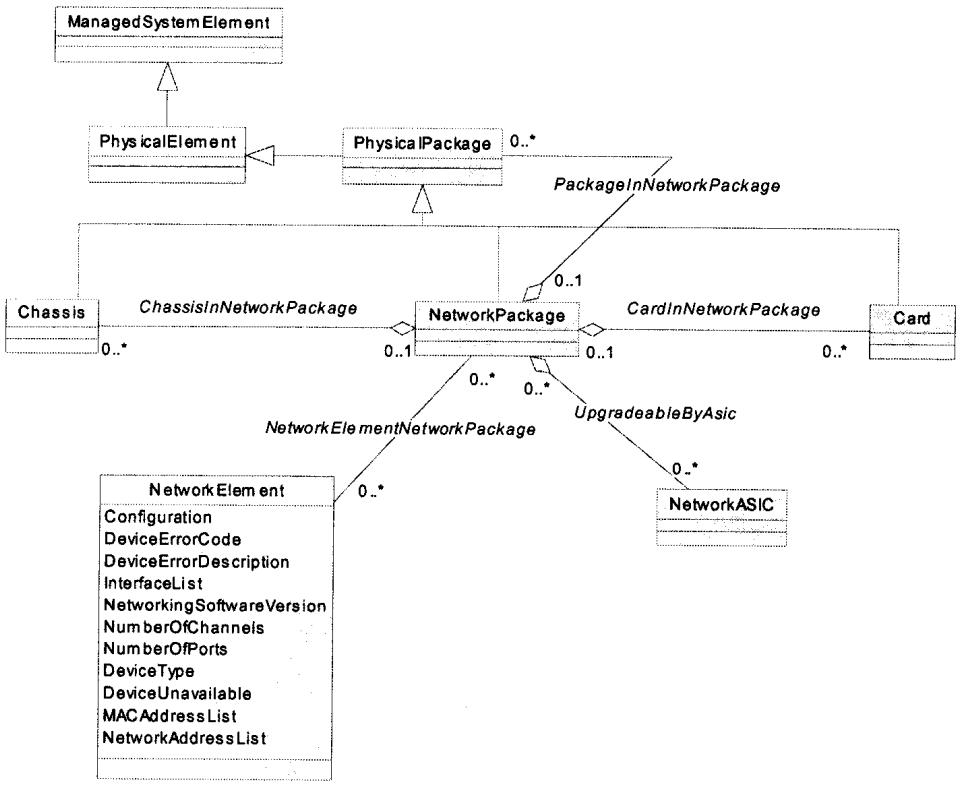

Figura a1.14 - Diagrama de Classes para Pacotes (Package) da Rede

8.1.9. Hierarquia da Classe NetworkElement

A porção lógica de elementos de rede é implementada criando-se uma nova classe base chamada NetworkElement definida como subclasse da classe CIM UnitaryComputerSystem. A aplicação é feita desta forma porque os dispositivos de rede mais complexos (ex.: roteadores e switches) possuem várias características de um sistema de computador descrito pela especificação CIM.

A classe NetworkElement adiciona funcionalidades específicas de rede aos conceitos básicos de um sistema de computador como: a habilidade de ter arquivos especiais de configuração que controlam a funcionalidade do dispositivo, atributos que descrevem a semântica do dispositivo e outros conceitos específicos de software de rede.

Funcionalidade específicas de elementos de rede também são adicionadas em forma de atributos pela classe NetworkElement.

A hierarquia especificada pelo DEN é mostrada na figura a1.15 abaixo:

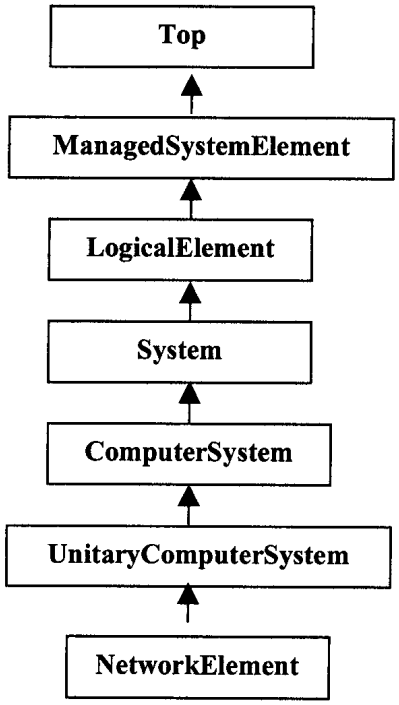

Figura a1.15 - Hierarquia da Classe NetworkElement

Diagrama de Classe para Elementos de Rede

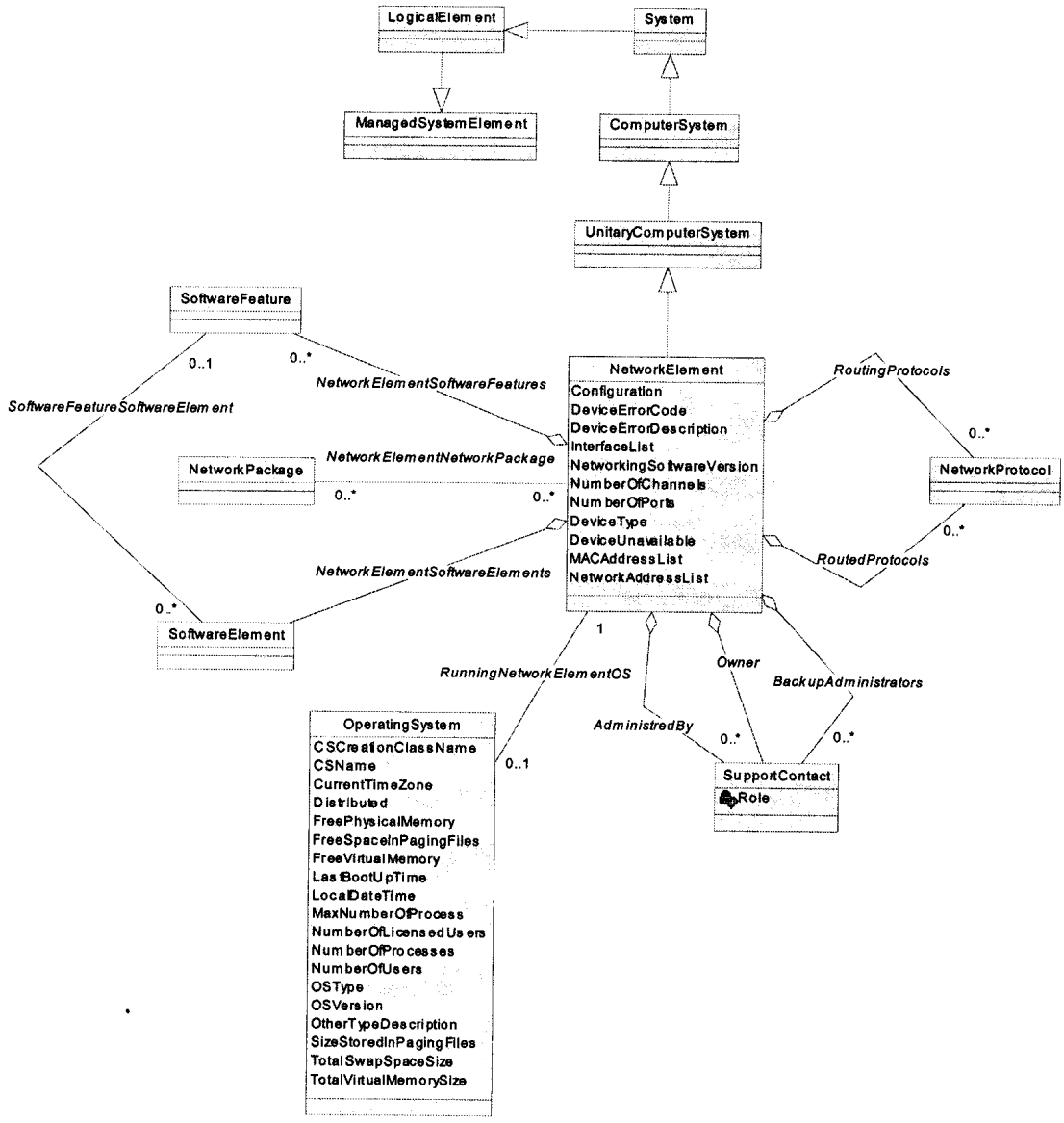

Figura a1.16 - Diagrama de Classes para Elementos de Rede

8.1.10. Hierarquia da Classe Network Media

Esta hierarquia de classe é definida pelo atual esquema CIM e é utilizada para representar medias físicas como fitas e discos rígidos removíveis. A especificação DEN propõe que esta classe seja renomeada para "PhysicalMedia"

e uma nova classe "NetworkMedia" seja introduzida para representar comunicações de rede.

A proposta para esta hierarquia de classe é demonstrada abaixo:

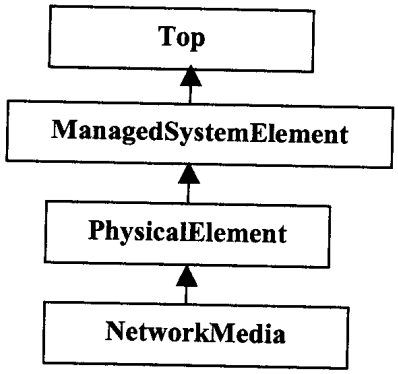

Figura a1.17 - Hierarquia da Classe NetworkMedia

Diagrama de Classe para Medias de Rede

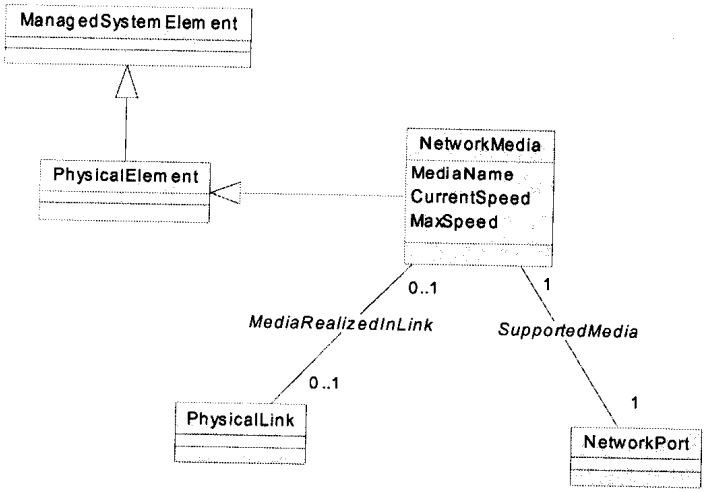

Figura a1.18 - Diagrama de Classes para Medias de Rede

8.1.11. Hierarquia da Classe Protocol

Esta hierarquia está sujeita a alterações devido a refinamentos no esquema CIM que podem ser realizados pelo grupo de trabalho de rede do DMTF. Nesta hierarquia são colocadas classes que agrupam características de diferentes protocolos de rede

Esta hierarquia de classes é representada abaixo:

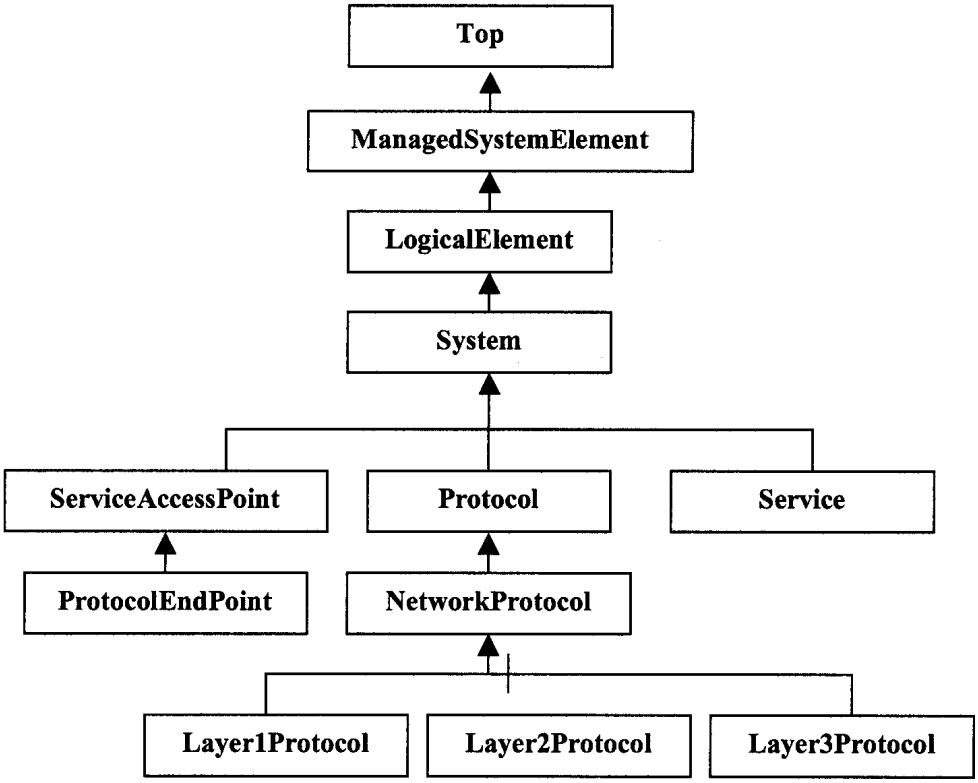

Figura a1.19 - Hierarquia da Classe Protocol

Diagrama de Classe para Protocolos

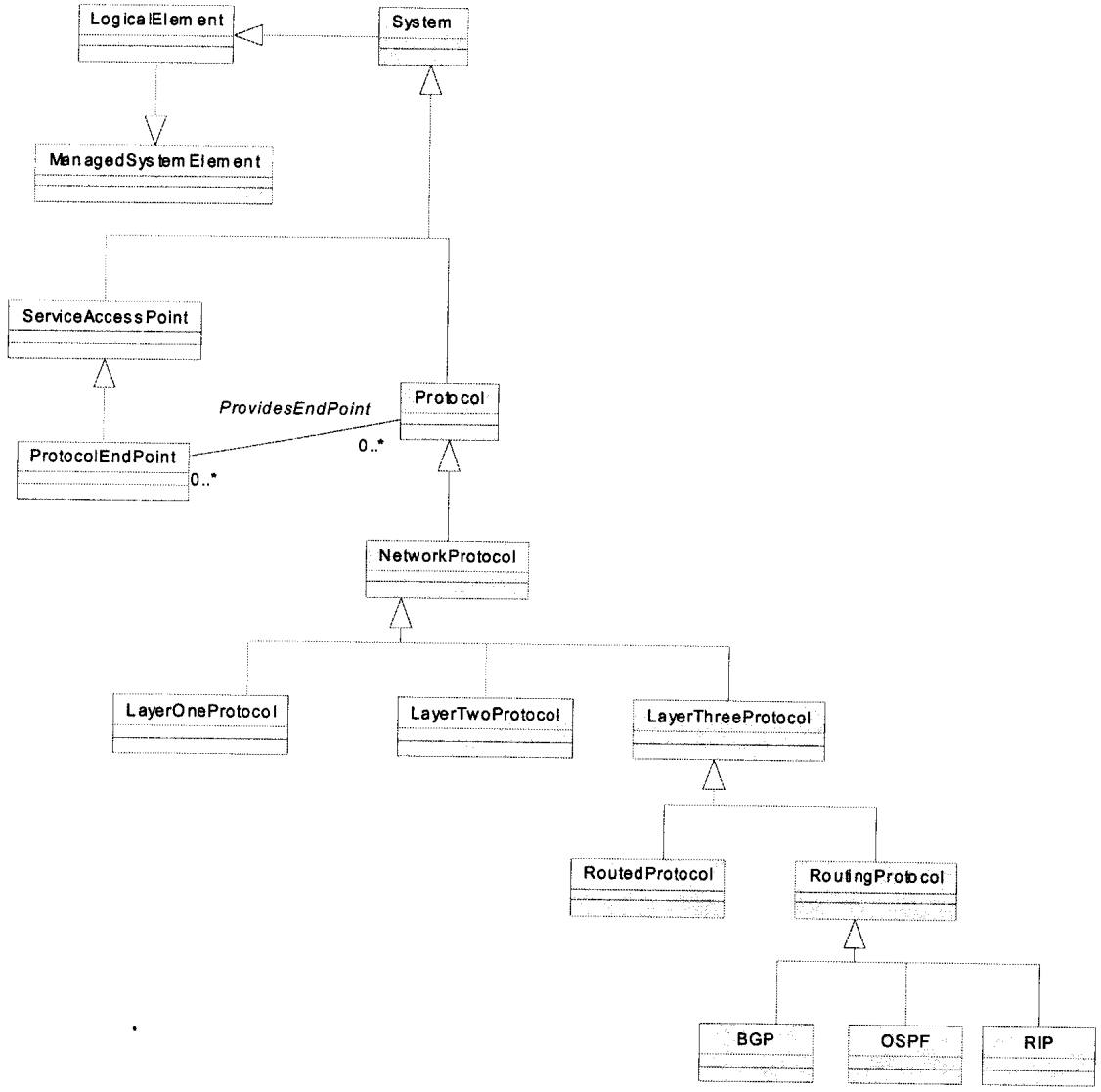

Figura a1.20 - Diagrama de Classes para Protocolos de Rede

8.1.12. Hierarquia da Classe User

As classes Person e OrganizationPerson são definidas pelo X.500. A especificação DEN adiciona propriedades chaves para permitir que serviços de rede sejam personalizados em base usuário por usuário.

As classes são as seguintes:

- Person: Classe X.500 utilizada para definir objetos que representam pessoas. A esta classe foram adicionados alguns atributos para permitir que pessoas estejam aptas a receber serviços de rede apropriados e também sejam limitadas por políticas apropriadas.

Person possui os seguintes relacionamentos:

- PersonRole: Associação zero-ou-mais para zero-ou-mais que relaciona uma pessoa a um ou mais papéis (roles) que esta pessoa possui.
- PersonContact: Associação zero-ou-mais para zero-ou-mais que relaciona funções de suporte a pessoas a produtos.

Person possui a seguinte subclasse:

- OrganizationPerson: Classe X.500 utilizada para definir objetos que representam empregados de uma determinada organização. A esta classe também foram adicionados alguns atributos para permitir que pessoas em uma determinada organização recebam serviços de rede apropriados e sejam limitados por política apropriadas.
- Organization: Classe X.500 que definem objetos que representam organizações. A esta classe foram adicionados alguns atributos para permitir que pessoas que pertencem a uma determinada organização recebam serviços de rede apropriados e sejam limitados por política apropriadas.

Serão definidas várias associações para Organization, incluindo:

- OrganizationRole: Associação zero-ou-mais para zero-ou-mais que relaciona uma organização com um ou mais papéis executados por esta organização.

- OrganizationContact: Associação zero-ou-mais para zero-ou-mais que relaciona funções de suporte a pessoas executados por uma organização a produtos.

- OrganizationUnit: Classe X.500 utilizada para definir objetos que representam porções de uma organização. A esta classe foram adicionados alguns atributos para permitir que pessoas que pertencem a uma determinada organização recebam serviços de rede apropriados e sejam limitados por política apropriadas.

Serão definidas várias associações para OrganizationUnit, incluindo:

- OrganizationUnitRole: Associação zero-ou-mais para zero-ou-mais que relaciona uma unidade organizacional a um ou mais papéis executados pela organização.
- OrganizationUnitContact: Associação zero-ou-mais para zero-ou-mais que relaciona funções de suporte a pessoas executados por uma unidade organizacional para produtos.

ANEXO 2 – CLASSES E ATRIBUTOS DO ESQUEMA RELEVANTES PARA A SEGURANÇA

As descrições das classes e principais atributos são as seguintes:

- LogicalElement

  Classe abstrata base para todos os componentes de um sistema que representa uma funcionalidade abstrata como: arquivos, processos, etc.

- OperatingSystem

  Representa um sistema operacional, definido como um software/firmware que faz com que um sistema computacional seja utilizável e implementa e/ou gerencia os recursos disponíveis neste sistema.

  Atributos:

  - CSCreationClassName : Indica o nome da classe ou subclasse utilizada para criar uma instância de OperatingSystem.
  - CSName : Nome do Sistema Operacional.
  - CurrentTimeZone : Define a TimeZone do sistema operacional.
  - Distributed : Verdadeiro (True) indica que o sistema operacional é distribuído através de vários nós (ComputerSystem).
  - FreePhysicalMemory : Número de Kbytes de memória física atualmente disponíveis.
  - FreeSpaceInPagingFiles : Número de Kbytes que podem ser mapeados no arquivo de paginação do sistema operacional sem causar swapp de páginas da memória.
  - FreeVirtualMemory : Número de Kbytes de memória virtual disponível.

- LastBootUpTime : Indica última inicialização (boot) do sistema operacional.
- LocalDateTime : Data e hora local.
- MaxNumberOfProcess : Número máximo de processos que o sistema operacional pode suportar.
- NumberOfLicensedUsers : Número de licenças de usuário para este sistema operacional.
- NumberOfProcesses : Número de processos atualmente sendo executados pelo sistema operacional.
- NumberOfUsers : Número de sessões de usuário atuais.
- OSType : Tipo do sistema operacional.
- OSVersion : Versão do sistema operacional.
- OtherTypeDescription : Descreve o fornecedor e o tipo do sistema operacional.
- SizeStoredInPagingFiles : Número total de Kbytes que pode ser armazenado no arquivo de paginação.
- TotalSwapSpaceSize : Número de Kbytes de memória suportado pelos processos de swapping.
- TotalVirtualMemorySize : Número de Kbytes de memória virtual suportado.

- Service

Elemento lógico que contém informações necessárias para representar e gerenciar a funcionalidade por um dispositivo ou características de software.

Atributos:

- ServiceCreationClassName : Fornece escopo para a hierarquia da classe Service.
- ServiceErrorCode : Fornece um código de erro para a falha do serviço.

- ServiceErrorDescription : Fornece informações detalhadas para o código de erro.
- ServiceKeywords : Lista de palavras chaves que ajuda na localização deste serviço.
- ServiceName : Nome amigável para este serviço.
- ServiceStartupConditions : Especifica as condições necessárias para a inicialização deste serviço. Atributo multi valorado.
- ServiceStartupParameters : Indica os parâmetros de inicialização que devem ser fornecidos para o startup deste serviço.
- ServiceURL : URL que fornece informações detalhadas sobre o serviço.
- ServiceUsage : Texto que descreve como se utilizar este serviço.
- ServicesContained : Conjunto de object references para um conjunto de objetos Service.
- Started : Verdadeiro (True) indica que este serviço foi inicializado.
- StartMode : Indica por quem este serviço deve ser inicializado (Ex.: sistema operacional, outro sistema, etc).

- AAAAService

Define objetos serviço que podem ser utilizados para implementar serviços de autorização, autenticação, contabilização e auditoria. É esperado que cada objeto AAAA será compreendido de um número de objetos que controlará um ou mais aspectos de todo o conjunto de serviços AAAA descritos pelo objeto AAAA pai.

Atributos:

- AAAAName : Nome amigável para esta classe AAAA.
- AAAAServicesContained : Lista de referência a objetos que definem serviços individuais agregados por esta classe AAAA.
- AAAAType : Tipo de serviço desta classe AAAA. Utilizado para categorizar diferentes tipos de classes AAAA.

- AuthenticationService

  Define objetos serviço que podem ser utilizados para prover autenticação de um dado objeto.

  Atributos:

  - AuthenticatonMethod : Define o tipo de autenticação a ser utilizada.
  - RequiresAuthentication : valor TRUE indica que este serviço requer autorização, mesmo depois da autenticação ter sido executada.

- Person

  Classe utilizada para definir o conceito genérico de pessoa.

  Atributos:

  - cn : Nome da pessoa
  - sn : Sobrenome da pessoa
  - description : Outras informações sobre a pessoa.
  - Observações: Observações sobre a pessoa.
  - seeAlso : Local onde pode ser buscada mais informações sobre a pessoa (Ex.: URL).
  - telephoneNumber : Número de telefone onde a pessoa pode ser encontrada.

- SOUser

  Classe utilizada para definir um susário cadastrado no sistema operacional.

  Atributos:

  - uid : Identificador único do usuário
  - userPassword : Senha de logon do usuário
  - userDomainID : Identificador do domínio administrativo do usuário
  - userAcctExpires : Identifica quando a conta do usuário expira.

- userBadPwCount : Número de vezes que o login do usuário falhou.
- userLastLogoff : Último logoff do usuário
- userLastLogon : Último logon do usuário
- userLogonHours : Horário que o usuário pode se conectar
- userLogonserver : Servidor que a requisição de logon do usuário deve ser enviada
- userNumLogons : Número de logons com sucesso realizados pelo usuário
- userPasswordExpired : Indica se a senha do usuário expirou.
- UserPriv : Nível de privilégio do usuário na rede.
- userScriptPath : Caminho onde se localiza o script de logon do usuário.
- userWorkstations : Workstations as quais o usuários pode se logar

- Profile

Define características e necessidades de um objeto. Por exemplo, um objeto UserProfile poderia definir como um usuário conecta-se a rede (características) e quais serviços estão disponíveis para aquele usuário uma vez que ele esteja conectado.

Atributos:

- NetworkClientServicesList : Conjunto de Object References para um conjunto de objetos Service Este atributo especifica os serviços a serem atribuídos em favor do usuário. Isto pode incluir o nome do servidor DNS primário e secundário a ser utilizado, o nome do domínio a ser utilizado e outros parâmetros que são utilizados para configurar um cliente. Este atributo é multi valorado.
- NetworkParametersList : Este atributo contém a lista de parâmetros que são utilizados para configurar a conexão do

usuário na rede. Exemplos incluem: quando pegar um endereço IP dinamicamente ou estaticamente, o nome do host, máscara de sub rede e gateway a ser atribuído a este usuário. Este atributo é multi valorado.

- ProfileName : Nome do Profile.
- ProfileType : Define o tipo de Profile desta instância. Utilizado para categorizar diferentes tipos de Profile.
- ServicesRequiredList : Conjunto de Object References para um conjunto de objetos Service. Este atributo especifica o conjunto de serviços que este objeto requer. Este atributo é multi valorado.
- ServicesReceivedList : Conjunto de Object References para um conjunto de objetos Service. Este atributo especifica o conjunto de serviços que este objeto recebeu (uma vez que tenha se conectado com sucesso). Este atributo é multi valorado.

- UserProfile

Define os serviços que o usuário está autorizado a utilizar. Também define se o usuário necessita ser autenticado antes do serviço ser concedido e que método de autenticação deve ser utilizado

Atributos:

- UserServicesList : Conjunto de Object References para um conjunto de objetos Service. Este atributo especifica o conjunto de serviços de rede, direitos e privilégios que o usuário tem acesso durante esta seção. este atributo é multi valorado.

- GroupProfile

Define os serviços que um grupo está autorizado a utilizar. Todo usuário que for membro deste grupo possui estes serviços por default.

Atributos:

- GroupServicesList : Conjunto de Object References para um conjunto de objetos Service. Membros deste grupo recebem este conjunto de serviços por default durante esta seção. Este atributo é multi valorado.

- PhysicalElement

  Classe utilizada para descrever qualquer componente físico do sistema.

- Top

  Classe Origem do diretório da qual todas as outras classes são derivadas. Esta classe foi buscada no esquema do X.500 e estendida para modelar elementos de rede.

- AuthorizationService

  Classe que define objetos serviço que podem ser utilizados para fornecer autorização para um dado objeto.

  Atributos:

  - AuthorizedServicesList : Conjunto de object references para um conjunto de objetos serviço. Lista que representa os serviços que um usuário autenticado pode utilizar. Atributo multi valorado.

- ACL

  Classe que identifica as ACLs dos objetos que devem sofrer autorização.

- ACE

  Classe que identifica as ACEs de uma dada ACL.

  Atributos:

  - aceSid : Identificador do usuário na ACE.
  - aceHeader : Identifica o tipo da ACE (access allowed, access denied, etc).
  - aceMask : Determina o que o usuário (sid) pode fazer com o objeto.
  - aceObjectName: Nome do objeto que sofre autorização.
  - aceObjectPath: Caminho utilizado para acessar o objeto (inclui seu nome).

- SOUserGroup

  Identifica os grupos de usuários definidos no sistema operacional.

  Atributos:

  - description: informações gerais para o grupo de usuários.

- SOSharedFile

  Identifica os arquivos compartilhados no sistema operacional. Está classe é implementada para que o diretório contenha apenas os dados dos arquivos compartilhados, pois apenas estes arquivos devem sofrer autorização.

  Atributos:

  - cn: nome do arquivo compartilhado (com caminho).

- SOSharedDirectory

  Identifica os diretórios compartilhados na rede. Esta classe é implementada para que o diretório contenha apenas os dados dos diretórios compartilhados, pois apenas estes devem sofrer autorização.

  Atributos:

  - cn: nome do diretório (com caminho)

- Policy

  Define um template para atributos e métodos que descrevem como funções podem ser invocadas para controlar como várias entidades interagem entre si.

  Atributos:

  - PoliciesContainedList : Especifica o conjunto de políticas individuais que são agregadas por esta política. Atributo multi valorado.
  - PolicyActionList : Conjunto de Object References para um conjunto de objetos PolicyAction. Representa uma ou mais ações que deverão ser executadas se a condição da política for satisfeita. Assume que as políticas devem ser executadas na ordem listada.
  - PolicyConditionList : Conjunto de ObjectReferences para um conjunto de objetos PolicyCondition. A agregação destas condições e as relações entre elas formam os requisitos que devem ser obedecidos para que as ações especificadas por esta política sejam executadas.
  - PolicyEnabled : TRUE indica que a política está ativa.
  - PolicyKind : Define se esta é uma política Service ou Usage.
  - PolicyModality : Especifica se as ações especificadas nesta política devem ser aplicadas se a condição é satisfeita (valor = 0) ou não satisfeita (valor = 1)
  - PolicyName : Nome amigável para esta classe de políticas.

- PolicyConditionRelations : Define como cada regra individual de política que é parte desta política estão relacionadas umas com as outras. Valores possíveis podem ser: =, !=, >, =>,<, =<, IN e NOT IN.
- PolicyUsage : Texto que descreve a forma de se utilizar esta política.
- PolicyErrorCode : Fornece um código de erro generalizado para indicar a falha da política.
- PolicyErrorDescription : Fornece uma descrição detalhada para uma propriedade PolicyErrorCode.
- PolicyKeywords : Lista de palavras chaves utilizadas para localizar esta política. Este atributo é multi valorado.
- PolicyType : Tipo de política a qual esta classe pertence. Este atributo é utilizado para categorizar diferentes tipos de políticas.

- PolicyAction

Classe utilizada para descrever o conjunto de ações a serem invocadas quando uma política é satisfeita. Se uma política requer múltiplas ações a serem executadas, então ela agregará vários objetos PolicyAction.

Atributos:

- ChangeAccessAction : Define uma alteração no privilégio de acesso para um trafego particular em um dispositivo. Note que não especifica o mecanismo (Ex.: ACL) pelo qual a alteração é feita.
- DSCodePointSetting : Um bit pattern específico do campo DS para implementar um comportamento Per-Hob (Ex.: Explicit Forwarding) como definido na arquitetura Differentiated Services.
- INProfileSetting : Define o setting do bit IN no byte DS da arquitetura Differentiated Services

- INProfileAction : Define a ação a ser tomada quando um pacote tem seu conjunto de bit IN/OUT. Ações incluem: None (significa que não há a necessidade de tráfego adicional para condicionamento) e Remark DS Field (altera o DS codepoint do pacote).
- OUTProfileAction : Define as ações a serem tomadas quando pacotes possuem seu bit OUT. Ações incluem: None (significa que não há mais a necessidade de condicionamento de tráfego), Delay (que atrasa os pacotes até que eles estejam in-profile), Discard e Remark (Ex.: alterar o campo DS para um novo DS CodePoint).
- TrafficAction : Define o tipo de ação a ser aplicada a este tipo de tráfego. Inclui: Permit, Deny, Encrypt, Start Accounting, Stop Accounting, Start auditing, Stop Auditing e Log.
- TrafficPrioritySetting : Define o valor relativo da prioridade indicado na propriedade TrafficPriorityType. Por exemplo, se a propriedade TrafficPriorityType é 1, então este campo marcará o campo de precedência de IP no cabeçalho IP para o valor definirá um comportamento para o gerenciamento de tráfego ou QoS.
- TrafficPriorityType : Define o tipo específico da prioridade relativa que será aplicada no dispositivo de rede. Valores incluem: O (None), 1 (IP Precedence), 2 (Token Ring), 3 (802.1p), 4 (ATM), 5 (Frame Relay), 6 (MPLS) e 7 (RSVP).
- TrafficQueueAssignment : Define a qual fila este tráfego deve ser atribuído. Note que ele não especifica o tipo da fila. Desta forma ela é independente de implementação e pode ser utilizada para atribuir tráfegos para diferentes tipos de filas(Ex.: class-based como priority-queue).
- TrafficQueueBehavior : Utilizado para atribuir um comportamento específico para condicionar o tráfego.

- PolicyCondition

  Classe utilizada para descrever o conjunto de condições que devem ser obedecidas para satisfazer uma política. Se uma política requer várias condições, então ela deve agregar vários objetos PolicyConditions.

  Atributos:

  - ApplicationNameCondition : Define o nome da aplicação a qual esta condição é aplicada. Certas aplicações utilizam múltiplas portas. Esta propriedade permite que as portas definidas por estas aplicações sejam agrupadas e tratadas como uma unidade.
  - DestinationDeviceCondition: Define o dispositivo destino que esta condição é aplicada. Esta é uma representação genérica que inclui nome DNS, host name, endereço IP, endereço MAC, ou qualquer outro mecanismo utilizado para identificar de forma única um dispositivo destino.
  - LocalityConditionDN : Conjunto de Object References para um conjunto de localizações as quais esta política é aplicada. Isto permite que o tráfego seja classificado baseado em onde ele está entrando ou saindo da rede. Este atributo é multi valorado.
  - PortNumberRangeCondition : Define a porta ou range de portas que esta condição se aplica. este atributo é multi valorado.
  - ProtocolConditionDN : Esta é uma referência ao objeto Protocol que define um nome ou número de protocolo como parte de uma condição de política.
  - SourceDeviceCondition : Define o dispositivo fonte que esta condição se aplica. Esta é uma representação genérica que inclui: nome DNS, host name, interface, endereço IP, endereço MAC, ou qualquer outro mecanismo utilizado para identificar de forma única um dispositivo fonte.
  - TimeOfDayValidityRef : Referência ao objeto TimeOfDayValidity que define quando esta condição é valida.

- UserConditionDN : Conjunto de Object References para um conjunto de usuários individuais aos quais esta política se aplica. Este atributo possui valores multi valorados.
- UserGroupConditionDN : Conjunto de Object References para um conjunto de grupos de usuários aos quais esta política se aplica. este atributo é multi valorado.
- OrganizationConditionDN : Conjunto de Object References para um conjunto de organizações às quais esta política se aplica. Este é um atributo multi valorado.
- OUConditionDN : Conjunto de Object References para um conjunto de Unidades Organizacionais às quais esta política se aplica. Este é um atributo multi valorado.

- NetworkingPolicy

  Classe utilizada para se definir várias políticas de rede (Ex.: Como condicionar o tráfego de acordo com um conjunto de comportamentos definidos).

  Atributos:

  - ActiveTCA : Nome do Traffic Conditioning Agreement ativo para este dispositivo
  - DSDomainName : Nome amigável que define um conjunto de dispositivos contíguos que conformam com a arquitetura Differentiated Services. Isto significa que eles operam em um serviço de provisioning policy comum e um conjunto de um grupo de definições PHB.
  - PHB : comportamento Per-Hope definido pela arquitetura Differentiated Services.
  - PHBGroup :

- SecurityPolicy

  Classe utilizada para definir várias políticas de segurança (Ex. IPSEC)

  Atributos:

  - SecurityID : Identifica de forma única cada instância de qualquer classe ou subclasse Security.
  - SecurityData : Dados contidos na SecurityPolicy
  - SecurityDataType : Identifica o tipo de dado que está contido em uma propriedade SecurityData
  - SecurityType : Especifica o tipo de serviço de segurança que este objeto é. Este atributo é utilizado para categorizar diferentes tipos de classes de segurança.

- UserGroupPolicy

  Classe utilizada para definir políticas baseadas em usuários que usam um grupo para agregar múltiplos usuários.

  Atributos:

  - UserGroupPoliciesActiveList : Define quais políticas devem ser ativadas para um grupo de usuários. Esta é uma função de como o usuário é conectado a rede quando este executa log in e outros fatores. Este valor é multi valorado.
  - UserGroupPoliciesList : Conjunto de Object References para um conjunto de objetos Policy. Estes objetos são utilizados para determinar quais serviços este grupo de usuários recebe quando um usuário executa log on ou requisita um serviço da rede. Membros deste grupo receber estas políticas por default. Este é um atributo multi valorado.

- ConfigurationPolicy

  Classe utilizada para se definir vários meios de se configurar dispositivos.

  Atributos:

  - ConfigurationMethod : Define como o dispositivo deve ser configurado. Valores incluem: 0 (Manualmente), 1 (Arquivo de Configuração), 2 (Arquivo de Configuração Alterado pelo Usuário), 3 (Outro), 4 (Unknown)

- DeviceGroupPolicy

  Classe utilizada para definir políticas baseadas em dispositivos utilizando um grupo para agregação. Cada dispositivo pertencente ao grupo recebe esta política por default.

  Atributos:

  - DeviceGroupPoliciesActiveList : Define quais políticas devem ser ativadas para um dado grupo de dispositivos. Esta é uma função do papel que o dispositivo tem na rede(Ex.: este é um dispositivo backbone ou é um dispositivo da extremidade) e outros fatores. Este atributo é multi valorado.
  - DeviceGroupPoliciesList : Conjunto de Object References para um conjunto de objetos Policy. Estes objetos são utilizados com os dispositivos para determinar quais políticas devem ser aplicadas a este grupo de dispositivos. Membros deste grupo recebem este conjunto de políticas por default quando são inicializados ou reconfigurados

- UserPolicy

  Classe utilizada para se definir várias políticas baseadas em usuários.

  Atributos:

  - UserPoliciesActiveList : Define quais políticas devem ser ativadas para um dado usuário. Esta é uma função de como o usuário é conectado a rede quando executa log in ou outros fatores.
  - UserPoliciesList : Conjunto de Object References para um conjunto de objetos Policy. Estes objetos são utilizados para determinar quais serviços este usuário receberá quando se logar ou requisitar serviços na rede. Este é um atributo multi valorado.

- NetworkElement

  Representa características físicas de dispositivos de rede

  Atributos:

  - Configuration : Contém as informações de configuração para este dispositivo.
  - DeviceErrorCode : Código de erro que indica a falha do dispositivo.
  - DeviceErrorDescription : Descrição detalhada para o código de erro.
  - InterfaceList : Identifica a lista de interfaces suportada pelo dispositivo. Este atributo é multi valorado. Valores podem ser ATM, Ethernet, Fast Ethernet, Token Ring, etc.
  - NetworkingSoftwareVersion : Indica a versão do software do dispositivo.
  - NumberOfChannels : Número de canais do dispositivo.
  - NumberOfPorts : Número de portas suportadas pelo dispositivo.
  - DeviceType : Define o tipo do dispositivo. Utilizado para categorizar dispositivos.

- DeviceUnavailable : Verdadeiro (True) indica que o dispositivo não está disponível.
- MACAddressList : Lista de endereços MAC neste dispositivo. Lista que pode conter um conjunto de endereços MAC, um para cada porta do dispositivo.
- NetworkAddressList : Lista de endereços de rede para o dispositivo.

- SupportContact

  Pessoa responsável pelo suporte.

  Atributos:

  - Role : Papel da pessoa de suporte. (Backup Operator, System Administrator, Database Administrator, etc).

ANEXO 3 – EXEMPLOS PARA ACESSO AO DIRETÓRIO UTILIZANDO LDAP

Para um melhor entendimento de como é realizado o acesso ao diretório, serão apresentados dois exemplos práticos que demonstram a utilização de funções de acesso ao repositório de dados do diretório.

O ambiente onde foi construído estes exemplos possui um serviço de diretório independente do sistema operacional, ou seja, o serviço de diretório não vem integrado ao sistema operacional. O serviço de diretório utilizado é o Netscape Directory Server 4.0 sobre o sistema operacional Microsoft Windows NT Server 4.0. Para construção dos programas foram utilizadas as linguagens Java, C e C++ juntamente com o SDK disponibilizado pela Netscape para acesso ao diretório. Todo o acesso é realizado utilizando uma API que implementa funções de acesso e manipulação das informações do diretório via LDAP.

Para execução destes programas exemplos é necessário que seja implementado os diagramas de classes de autenticação e autorização apresentados por este trabalho.

Serão apresentados dois exemplos. O primeiro exemplo, demonstra a utilização da autenticação de usuários através do diretório. Este exemplo demonstra como pode ser realizada implementações de aplicações que utilizem o serviço de diretório como repositório de usuários que possam ser autenticados para utilizar estas aplicações. Para execução deste exemplo, após implementado o diagrama de classes de autenticação, basta cadastrar os usuários que utilizaram a aplicação na classe SOUser. Esta classe armazenará, entre outros, o nome do usuário e sua senha devidamente criptografada. O código do programa foi escrito utilizando-se a linguagem Java e o SDK Java da Netscape para acesso ao diretório.

O código é o seguinte:

```
package Seguranca;


import netscape.ldap.*;
/*
* Este programa autentica usuários através do serviço de diretório
* Uso: autenticaUsuario uid senha
*/
public class autenticaUsuario{
  public static boolean autentica(String uid, String pwd)
  {
    boolean status = false;
    LDAPConnection ld = new LDAPConnection();
    LDAPEntry findEntry = null;
    String dn = null;
    String MY_HOST = "localhost";
    int MY_PORT = 389;
    String MY_SEARCHBASE = "o=casa.com.br";
    String MY_FILTER = "uid=" + uid;

   try{
     ld.connect(MY_HOST, MY_PORT);
LDAPSearchResults res = ld.search(MY_SEARCHBASE,
 LDAPConnection.SCOPE_SUB,MY_FILTER,null,false);

if (res.hasMoreElements()){
 findEntry = res.next();
 dn = findEntry.getDN();
          .
      System.out.println("dn is "+ dn);


 //Previne conexoes anonimas
 if ((dn == "") || (pwd == "")) {return false;}
      //realiza bind com o servidor
 ld.authenticate(dn, pwd);

      //Se ld.authenticate nao sofre exeção
 status = true;
   }
 }
 catch (LDAPException e) {
    System.out.println(e.toString());
 }
 catch (Exception x) {
```

```
            x.printStackTrace();
        }
        return status;
    }

    public static void main (String args[]){

        if (args.length < 2) {
            System.out.println("Uso: ldapAuth userid senha");
            System.exit(1);
        }

        String uid = args[0];
        String passwd = args[1];
        boolean status = autentica(uid, passwd);
        if (status) {
            System.out.println(uid + " autenticado!");
        }
        else {
            System.out.println(uid + " falhou!");
        }
        System.exit(0);
    }
}
```

O segundo exemplo demonstra como pode ser utilizado o serviço de diretório para armazenar de forma centralizada as permissões de acesso de objetos do ambiente. No exemplo, foi implementado um programa que dá permissões em arquivos ou diretórios do sistema operacional a partir de informações buscadas na classe ACE (access control entry) do esquema proposto. Para execução deste programa, basta cadastrar as ACEs no repositório de dados do diretório informando qual arquivo ou diretório alvo, qual o usuário que deve receber a informação, qual a acess mask (nível de acesso) do usuário. Implementações mais complexas poderão utilizar os demais atributos desta classe como por exemplo o atributo aceHeader onde pode ser implementado ACEs que permitirão acesso (access allowed) e ACEs que restringirão o acesso (access denied). O código do programa foi escrito na linguagem C++ utilizando-se a API C disponibilizada pela Netscape através do SDK C para acesso ao serviço de diretório.

O código é o seguinte:

Listagem do arquivo de cabeçalho:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
//#include <afx.h>


//includes para alteraAcl
#include <windows.h>
#include <iostream.h>

#include <ldap.h>

/*
 * Host name of LDAP server
 */
#define MY_HOST          "localhost"

/*
 * Port number where LDAP server is running
 */
#define       MY_PORT          LDAP_PORT

/*
 * DN of directory manager entry.  This entry should have write access to
 * the entire directory.
 */
#define MGR_DN           "cn=Directory Manager"

/*
 * Password for manager DN.
 */
#define MGR_PW           "pmanager"

/*
 * Subtree to search
 */
#define       MY_SEARCHBASE       "o=casa.com.br"

/*
 * Place where people entries are stored
```

```
 */
#define PEOPLE_BASE    "ou=Central, " MY_SEARCHBASE

/*
 * Filter to use when searching.  This one searches for all entries with the
 * surname (last name) of "Jensen".
 */
//#define       MY_FILTER "(sn=Jensen)"

/*
 * Entry to retrieve
 */
#define ENTRYDN PEOPLE_BASE
Listagem do programa principal:
*
 *
 * Busca os atributos de uma ACE.
 */

#include "examples.h"

void alteraAcl(LPTSTR aceSid, LPTSTR aceHeader, LPTSTR aceMask,
LPTSTR aceObjectPath);

int
main( int argc, char **argv )
{
   LDAP                 *ld;
   LDAPMessage    *result, *e;
   char         **vals, *attrs[ 5 ];
   int                  i; //entrada;

 //Variaveis para passagem de paramentros
 LPTSTR aceSid, aceHeader, aceMask, aceObjectName,
                    aceObjectPath;

   /* Busca um handle para a conexao LDAP */
  if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL )
  {
perror( "ldap_init" );
return( 1 );
  }
  attrs[ 0 ] = "aceSid";              /* Pega o identificador do usuário */
  attrs[ 1 ] = "aceHeader";           /* Pega o aceHeader */
  attrs[ 2 ] = "aceMask";                    /* Pega o aceMask*/
  attrs[ 3 ] = "aceObjectName";       /* Pega o nome do objeto */
  attrs[ 4 ] = "aceObjectPath";       /*Pega o caminho do arquivo.*/
```

```
  attrs[ 5 ] = NULL;

  if ( ldap_search_ext_s( ld, ENTRYDN,
    LDAP_SCOPE_SUBTREE,"(aceid=*)", attrs, 0,
   NULL,NULL,NULL,LDAP_NO_LIMIT,&result ) != LDAP_SUCCESS )
  {
ldap_perror( ld, "ldap_search_s" );
return( 1 );
  }

  int num_entries = ldap_count_entries( ld, result );
  cout << endl << "Existem " << num_entries << " entradas de ACEs no
      diretorio." << endl;
  cout << "---------------------------------------" << endl;

   /* Imprime as entradas */
  for ( e = ldap_first_entry( ld, result ); e != NULL;
    e = ldap_next_entry( ld, e ) )
  {

ldap_value_free( vals );


cout << "Dados da ACE:" << endl << endl;
if (( vals = ldap_get_values( ld, e, "acesid" )) != NULL )
{
        cout << "aceSid: ";
        for ( i = 0; vals[i] != NULL; i++ )
        {
                cout << vals[i] << endl;
                aceSid = vals[i];
        }

}
if (( vals = ldap_get_values( ld, e, "aceHeader" )) != NULL )
{
        cout << "aceHeader: ";
        for ( i = 0; vals[i] != NULL; i++ )
{
                cout << vals[i] << endl;
                aceHeader = vals[i];
}

  }
  if (( vals = ldap_get_values( ld, e, "aceMask" )) != NULL )
  {
cout << "aceMask: ";
```

```
  for ( i = 0; vals[i] != NULL; i++ )
   {
  cout << vals[i] << endl;
  aceMask = vals[i];
   }

   }
   if (( vals = ldap_get_values( ld, e, "aceObjectName" )) != NULL )
   {
  cout << "aceObjectName: ";
  for ( i = 0; vals[i] != NULL; i++ )
          {
          cout << vals[i] << endl;
          aceObjectName = vals[i];
  }

   }
   if (( vals = ldap_get_values( ld, e, "aceObjectPath" )) != NULL )
   {
  cout << "aceObjectPath: " ;
  for ( i = 0; vals[i] != NULL; i++ )
  {
          cout << vals[i] << endl;
          aceObjectPath = vals[i];
  }

   }
   //Chama a função de alteração de ACL para o arquivo/diretório
   //especificado em aceObjectPath para o usuario definido por aceSid
   //alteraAcl(aceSid, aceHeader, aceMask, aceObjectPath);

cout << endl << "---------------------------------------" << endl;
   }
   ldap_msgfree( result );
   ldap_unbind( ld );
   return( 0 );
}
//Função que altera as permissões no sistema de arquivos

void alteraAcl(LPTSTR aceSid, LPTSTR aceHeader, LPTSTR aceMask,
    LPTSTR aceObjectPath)
{
 BOOL ret;
 LONG err;
 SECURITY_DESCRIPTOR *sdData;
 SECURITY_DESCRIPTOR absSD;
 PSID psid;
```

```
PACL pacl;
PACL pNewACL;
DWORD newACLSize;
BOOL byDef;
BOOL haveDACL;
SID_NAME_USE sidType;
DWORD sidSize;
UINT x;
ACL_SIZE_INFORMATION aclSize;
ACCESS_ALLOWED_ACE *pace;
CHAR str[80];
DWORD strSize;
DWORD sizeRqd;

boolean bAlterou = FALSE;

// --- Busca o novo SID do grupo ---

//lookup SID do nome
sidSize=0;
strSize=80;
ret=LookupAccountName(NULL, aceSid, NULL, &sidSize, str,
        &strSize, &sidType);

if(sidSize)
{
        //aloca memoria para o SID
        psid=(PSID) GlobalAlloc(GPTR, sidSize);
        //pega SID
        strSize=80;
        ret=LookupAccountName(NULL, aceSid, psid, &sidSize,
                str, &strSize, &sidType);
        if (!ret)
        {
                cerr << "Nao foi possivel buscar o SID do usuario."
 << endl;
                return;
        }
}
else
{
        cerr << "O SID do usuario nao esta disponivel- ." << endl;
        return;
}

//---Busca uma copia do SD/DACL ---
```

```
// Verifica o quanto de memória é necessário
// para armazenar o SD w/DACL existente
sizeRqd=0;

err=GetFileSecurity(aceObjectPath,
          DACL_SECURITY_INFORMATION,NULL, sizeRqd, &sizeRqd);
if(err == ERROR_INSUFFICIENT_BUFFER)
{
        cerr << "Nao foi possivel buscar o tamanho do SID.:" <<
endl;
        return;
}

// Aloca a memória
sdData=(SECURITY_DESCRIPTOR *) GlobalAlloc(GPTR,
  sizeRqd);
if (sdData == NULL)
{
        cerr << "Não foi possível alocar memória." << endl;
        return;
}

// Pega a informação do SID atual

err=GetFileSecurity(aceObjectPath,
  DACL_SECURITY_INFORMATION,
   sdData, sizeRqd,&sizeRqd);
if (err == ERROR_SUCCESS)
{
        cerr << "Nao foi possavel pegar as informacoes do SID." <<
                endl;
        int num = GetLastError() ;
        if (num == 2)
                cout << "O arquivo especificado no atributo
  aceObjectPath na ACE nao pode ser encontrado."
  << endl;
        return;
}
// Cria um novo SD e DACL absoluto

//Inicializa o SD absoluto
ret=InitializeSecurityDescriptor(&absSD,SECURITY_DESCRIPTO
  R_REVISION);
if (!ret)
{
        cerr << "Não foi possível inicializar o novo SD." << endl;
        return;
```

```
}

// Busca informação do DACL
ret=GetSecurityDescriptorDacl(sdData, &haveDACL, &pacl,
        &byDef);
if (!ret)
{
        cerr << "Não foi possível buscar informações do DACL."<<
  endl;
        return;
}

if (!haveDACL)
{
        //computa o tamanho da nova DACL
        newACLSize = sizeof(ACCESS_ALLOWED_ACE) +
                GetLengthSid(psid) - sizeof(DWORD);
}
else
{
        //Busca a informação da DACL atual
        ret=GetAclInformation(pacl, &aclSize,
  sizeof(ACL_SIZE_INFORMATION), AclSizeInformation);

        //Computa o tamanho da nova DACL
        newACLSize=aclSize.AclBytesInUse +
  sizeof(ACCESS_ALLOWED_ACE) +
                GetLengthSid(psid) - sizeof(DWORD);
}

// Aloca Memória
pNewACL=(PACL) GlobalAlloc(GPTR,newACLSize);
if(pNewACL == NULL)
{
        cerr << "Não foi possível alocar memória." << endl;
        return;
}

//inicializa a nova DACL
ret=InitializeAcl(pNewACL, newACLSize, ACL_REVISION);
if (!ret)
{
        cerr << "Não foi possível inicializar a nova DACL."<< endl;
        return;
}
// Copia a DACL existente para a nova DACL
if(haveDACL)
```

```
{
        //copia as ACEs para a nova DACL
        for (x=0; x<aclSize.AceCount; x++)
        {
                ret=GetAce(pacl, x, (LPVOID *) &pace);
                if(!ret)
                {
                        cerr << "Nao foi possivel buscar a ACE." <<
 endl;
                        return;
                }

                ret=AddAce(pNewACL, ACL_REVISION,
 MAXDWORD, pace, pace->Header.AceSize);
                if(!ret)
                {
                        cerr << " Nao foi possivel adicionar a ACE."<<
 <<endl;
                        return;
                }
        }
}

//---Adiciona a nova ACE a nova DACL---

//Adiciona a ACE access allowed a nova DACL
//Verifica qual a Access Mask Permitida
int val = strcmp(aceMask,"full");
if (val == 0)
{
        ret=AddAccessAllowedAce(pNewACL, ACL_REVISION,
 GENERIC_ALL, psid);
        bAlterou = TRUE;
}

val = strcmp(aceMask,"execute");
if(val == 0)
{
        ret=AddAccessAllowedAce(pNewACL, ACL_REVISION,
 GENERIC_EXECUTE, psid);
        bAlterou = TRUE;
}
val = strcmp(aceMask,"write");
if (val == 0)
{
        ret=AddAccessAllowedAce(pNewACL, ACL_REVISION,
 GENERIC_WRITE, psid);
```

```
        bAlterou = TRUE;
}
val = strcmp(aceMask,"read");
if (val == 0)
{
        ret=AddAccessAllowedAce(pNewACL, ACL_REVISION,
  GENERIC_READ, psid);
        bAlterou = TRUE;
}

if(!ret)
{
        cerr << "Nao foi possivel adicionar a ACE."<< endl;
        return;
}

//Atribui a nova DACL no SD absoluto
ret=SetSecurityDescriptorDacl(&absSD, TRUE, pNewACL,
  FALSE);

if(!ret)
{
        cerr << "Nao foi possivel intalar a DACL."<< endl;
        return;
}

//Checa o novo SD
ret=IsValidSecurityDescriptor(&absSD);
if(!ret)
{
        cerr << "SD invalido."<< endl;
        return;
}

//---Instala a nova DACL---
//instala o SD alterado
err=SetFileSecurity(aceObjectPath,
  DACL_SECURITY_INFORMATION, &absSD);
if(err == ERROR_SUCCESS)
{
        cerr << "Nao foi possível atribui o SD ao
  arquivo/diretorio."<< endl;
        return;
}

// Garante que o SD tenha sido alterado
//RegCloseKey(HKEY_LOCAL_MACHINE);
```

```
// Libera a memoria
if(GlobalFree(pNewACL))
{

        cerr << "Nao foi possivel liberar memoria."<< endl;
        return;
}

// Libera Memoria
if(GlobalFree(psid))
{
        cerr << "Nao foi possivel liberar memoria."<< endl;
        return;
}

//Libera memoria
if(GlobalFree(sdData))
{
        cerr << "Nao foi possivel liberar memoria."<< endl;
        return;
}

if (bAlterou)
        cout << endl << "Permissao atribuida" << endl;
else
        cout << endl << "Nao foi possivel atribuir a permissao
  definida para a aceMask. Verifique os dados no
  diretorio.";

}
```

ANEXO 4 – DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência faz parte dos diagramas de interação definidos pela UML (Unified Modeling Language). Os diagramas de interação são modelos que descrevem como grupos de objetos colaboram a favor de um determinado comportamento. O diagrama de interação mostra um certo número de objetos e as mensagens que são passadas entre estes objetos.

Abaixo é mostrado um diagrama de sequência e seus principais componentes:

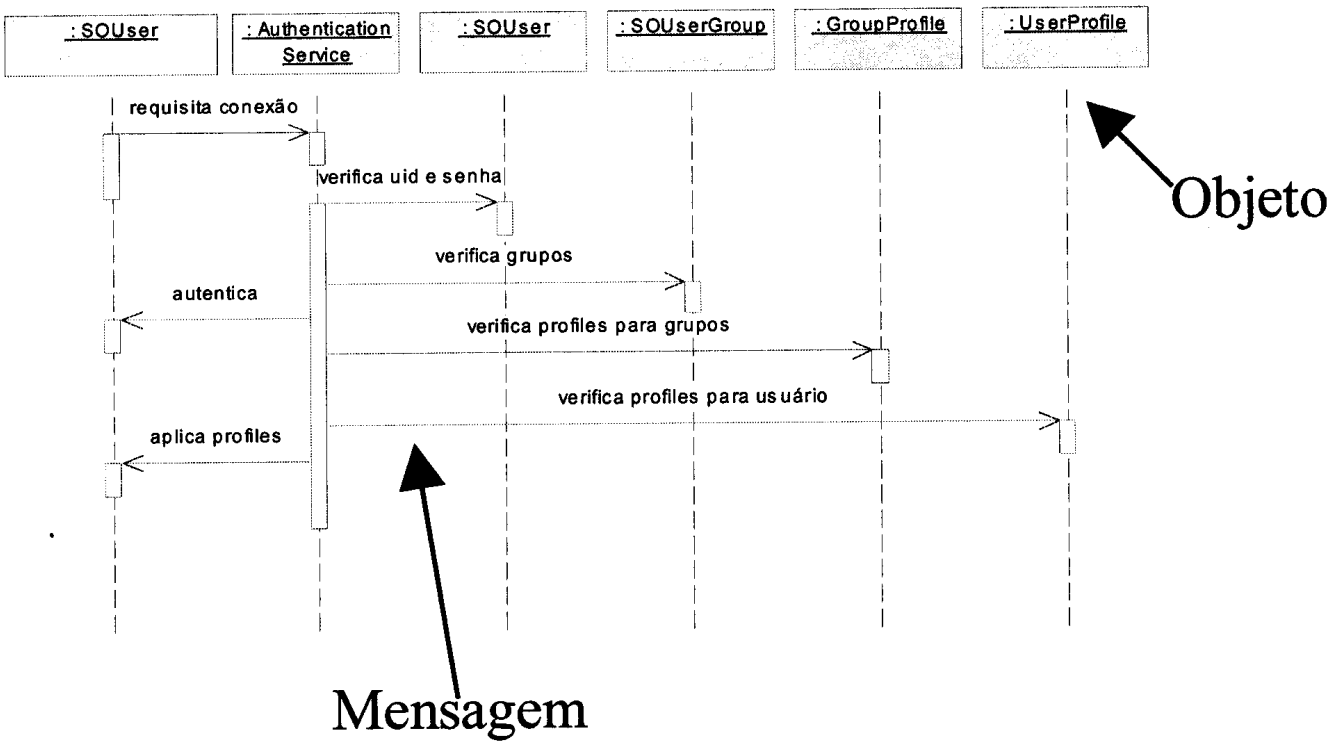

Figura a4.1 - Hierarquia da Classe Protocol

Os objetos ficam na parte superior do diagrama e as mensagens entre os objetos são as setas colocadas na horizontal.

Para um detalhamento destes e de outros componentes de um diagrama de sequência veja [FOW97].