Tiago Marchetti Dolphine

Plataforma de Serviços de Infra-estrutura para Arquiteturas de Mobilidade

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Orientador: Eleri Cardozo

Campinas, SP
2009

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

D698p

Dolphine, Tiago Marchetti
Plataforma de serviços de infra-estrutura para arquiteturas de mobilidade / Tiago Marchetti Dolphine. –Campinas, SP: [s.n.], 2009.

Orientador: Eleri Cardozo.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computação - Protocolos. 2. TCP/IP (Protocolo de rede de computação). 3. Telecomunicações - Tráfego. 4. Intenet (redes de computação). 5. Sistema de comunicação sem fio. I. Cardozo, Eleri. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês:	Infrastructure services platform for mobility architectures
Palavras-chave em Inglês:	Computer network protocols, TCP/IP (Computer network protocol), Telecommunications traffic, Internet (Computer: network), Wireless communication systems
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca examinadora:	Eleri Cardozo, Cesar Augusto Cavalheiro Marcondes, Fábio Luciano Verdi
Data da defesa:	27/08/2009
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Tiago Marchetti Dolphine

Data da Defesa: 27 de agosto de 2009

Título da Tese: "Plataforma de Serviços de Infra-estrutura para Arquiteturas de Mobilidade"

Prof. Dr. Eleri Cardozo (Presidente): _____

Prof. Dr. Cesar Augusto Cavalheiro Marcondes: _____

Prof. Dr. Fábio Luciano Verdi: _____

Resumo

Arquiteturas de micromobilidade vêm sendo propostas para atender ao crescente interesse por tecnologias de mobilidade IP. MPA (*Mobility Plane Architecture*), desenvolvida na FEEC/Unicamp, é uma solução de micromobilidade baseada em tunelamento de pacotes que emprega apenas protocolos bem estabelecidos. Quando estudou-se aplicações para melhoria de desempenho em *handover* e engenharia de tráfego na arquitetura MPA, verificou-se que estas aplicações necessitavam serviços comuns. Este trabalho descreve a plataforma MIS (*Mobility Infrastructure Services*) que fornece um conjunto de serviços para suporte ás funcionalidades básicas comuns de aplicações de gerência de rede, engenharia de tráfego e gerência de mobilidade. A plataforma proposta facilita o desenvolvimento de tais aplicações compartilhando soluções às necessidades encontradas na arquitetura de micromobilidade MPA e outras arquiteturas similares. São apresentados o projeto, implementação e testes da plataforma, juntamente com dois estudos de casos em gerência de mobilidade e engenharia de tráfego.

Palavras-chave: redes IP móveis, micromobilidade, gerência de rede, engenharia de tráfego.

Abstract

Micro-mobility architectures have been proposed to meet the growing interest in technologies supporting mobility in IP networks. MPA (Mobility Plane Architecture), developed at the FEEC/Unicamp, is a micro-mobility solution based on tunneling of packets that employs only well standardized protocols. When applications for supporting seamless handover and traffic engineering in MPA started to be designed, it was noticed that these applications demand a set of common services. This dissertation presents the MIS (Mobility Infrastructure Services) platform which provides a set of services that meet the common basic features demanded by applications of network management, traffic engineering and mobility management. The proposed platform facilitates the development of these applications when they are targeted to MPA and other micro-mobility architectures. This dissertation reports the design, implementation, ant test of the MIS platform, along with two case studies in mobility management and traffic engineering.

Keywords: IP mobile networks, micro-mobility, network management, traffic engineering.

Agradecimentos

Agradeço a Deus pela força, benção e motivação durante todo o período de pesquisa e elaboração deste trabalho.

Agradeço a minha família, em especial meu pai José Adilson Dolphine, minha mãe Solange Aparecida Marchetti Dolphine, minhas irmãs Larissa e Priscila e, minha avó Florinda, por todo apoio, amor, carinho e confiança imprescindíveis em minha vida.

Agradeço a minha namorada Maria de Lourdes Regina Gomes por toda dedicação, força e amor em todos os momentos.

Agradeço o meu orientador, professor Dr. Eleri Cardozo, por ter me orientado com extrema dedicação durante todo o período deste trabalho de Mestrado.

Agradeço os meus amigos Guilherme Patrezze, Fábio Campos, Ricardo Alberti, Iuri Diniz por todos os momentos de descontração e amizade incontestável.

Agradeço todos os colegas de trabalho do projeto MPA, em especial Rodrigo Prado por toda parceria, ajuda, sugestões e amizade dedicados, que tornaram possível e inspiraram este trabalho.

Agradeço aos meus amigos companheiros de república, Eder Ignatowicz , Liniquer Vieira , Daniel Sundfeld pela convivência, amizade e parceria em nossa residência.

Finalizando, agradeço a todos os amigos, colegas e familiares não mencionados mas que colaboraram com minha formação pessoal e profissional, que de certa forma contribuiriam para que este trabalho se concretizasse.

Dedico aos meus pais: Adilson e Solange, as minhas irmãs: Priscila e Larissa, a minha avó Florinda e a todos que acreditam em meu trabalho

Sumário

Lista de Figuras

Glossário

AJAX - Asynchronous Javascript And XML

API - Application Programming Interface

CoA - Care of Address

CoS - Class of Service

DHCP - Dynamic Host Configuration Protocol

DiffServ - Differentiated Services Architecture

FA - Foreign Agent

FMIPv6 - Fast Handovers for Mobile IPv6

HA - Home Agent

HI - Host Identifier

HIP - Host Identity Protocol

HMIPv6 - Hierarchical Mobile IPv6

HTTP - Hypertext Transfer Protocol

IETF - Internet Engineering Task Force

JAXB - Java Architecture for XML Binding

JPA - Java Persistence API

JSF - JavaServer Faces

LCoA - Local CoA

LMA - Local Mobility Anchor

MAG - Mobile Access Gateway

MAP - Mobility Anchor Point

MAR - Mobility Aware Router

MIB - Management Information Base

MIP - Mobile IP

MIPv4 - Mobile IPv4

MIPv6 - Mobile IPv6

MIS - Mobility Infrastructure Services

MPA - Mobility Plane Architecture

MPLS - Multiprotocol Label Switching

MVC - Model View Controller

P2MP - Point to Multipoint

PMIPv6 - Proxy Mobile IPv6

QoS - Quality of Service

RCoA - Regional CoA

REST - Representational state transfer

SNMP - Simple Network Management Protocol

SSH - Secure Shell

UML - Unified Modelling Language

URI - Universal Resource Identifier

XML - Extensible Markup Language

Trabalhos Publicados Pelo Autor

1. Tiago Dolphine, André Berenguel, Rodrigo Prado e Eleri Cardozo. "Plataforma de serviços de infra-estrutura para gerência de mobilidade". XIV Workshop de Gerência e Operação de Redes e Serviços - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos 2009, Recife, PE, Maio 2009.

2. Zagari, E., Prado, R., Badan, T., Cardozo, E., Magalhães, M., Carrilho, J., Berenguel, A., Moraes, D., Dolphine, T., Johnson, T., & Westberg, L. (2009). "Design and Implementation of a Network-Centric Micro-Mobility Architecture". IEEE Wireless Communications and Networking Conference, Budapeste, Hungria, Abril 2009.

Capítulo 1

Introdução

A crescente difusão da mobilidade verificada nos últimos anos se deve ao barateamento dos dispositivos móveis e aos avanços tecnológicos que permitiram a miniaturização desses dispositivos e um menor consumo de energia. Como conseqüência, observou-se um aumento do interesse por novas tecnologias de mobilidade no âmbito das redes TCP/IP.

Na solução de mobilidade *Mobile Internet Protocol* (MIP) [1] [2], proposta pela IETF (Internet *Engineering Task Force*), um nó móvel[1] sempre mantém a conectividade ao se deslocar de sua rede de origem graças ao mapeamento entre o endereço fixo do nó móvel e um endereço temporário obtido na rede visitada. Contudo, a conectividade do nó móvel fica comprometida devido ao intervalo de tempo gasto na sinalização cada vez que o nó móvel obtém um novo endereço temporário. Para amenizar este problema foram propostas algumas soluções de micromobilidade para reduzir o atraso e a perda de pacotes quando ocorrer um *handover*[2]. Soluções de micromobilidade procuram evitar que o nó móvel tenha que obter um novo endereço ao se deslocar entre pontos de conexão de um mesmo domínio ou criar uma hierarquia de mapeamentos que evite a necessidade de propagação da sinalização até a rede de origem.

MPA (*Mobility Plane Architecture*) [3] é uma arquitetura de micromobilidade desenvolvida no âmbito da FEEC/Unicamp. Essa arquitetura é baseada no estabelecimento de uma rede de sobreposição que utiliza túneis IP/IP ou MPLS (*Multiprotocol Label Switching*). A solução é centralizada na rede, ou seja, não exige atualização ou software adicional em nós móveis para suportar a mobilidade. MPA tem por objetivo manter a melhor performance durante o processo de *handover* visando diminuir a quantidade de informação perdida, de forma a minimizar a interferência para as aplicações de rede.

1.1 Motivações

A arquitetura MPA apresenta uma solução para micromobilidade que, embora apresente resultados que satisfazem as necessidades de diversas aplicações, poderia ter seu desempenho melhorado no sentido de solucionar problemas de congestionamento e desempenho no processo de *handover*. Tais melhorias de desempenho visam atender requisitos de aplicações que exigem baixa perda de

[1] Dispositivo que pode se movimentar e mudar seu ponto de conexão à rede.

[2] Troca de ponto de conexão à rede.

pacotes durante a comunicação, como, por exemplo, aplicações de áudio e vídeo em tempo real. Dessa maneira, foram conduzidos estudos na área de engenharia de tráfego e gerência de *handover* com objetivo de propor novos algoritmos e componentes para melhorar o desempenho da arquitetura MPA.

Em um estudo visando melhorias no processo de *handover* na arquitetura MPA [4] foram analisados processos de antecipação de *handover* para prover ações que visam a diminuição na perda de pacotes que são destinados ao nó móvel. Neste estudo foram observadas necessidades específicas de algumas funcionalidades como histórico de conexões de nó móvel, difusão destes eventos de conexão e atuação em recursos da rede.

Outro estudo com relação à engenharia de tráfego no controle de congestionamento foi conduzido visando melhorar o desempenho conforme enlaces vão se tornando saturados. A obtenção de dados de recursos da rede para monitoramento e atuação sob os mesmos mostrou-se necessária nesse estudo.

Diante das necessidades encontradas nos estudos realizados na arquitetura MPA tornou-se evidente que existiam diversas funcionalidades comuns e que estas acabariam sendo implementadas independentemente para cada uma das aplicações. Desta forma, cada aplicação incorporada à arquitetura MPA demandaria esforço de implementação dessas funcionalidades além de torná-las específicas para cada aplicação. Uma solução para estes problemas é a definição de uma plataforma única que desse suporte às funcionalidades comuns requeridas pelas aplicações de gerência de mobilidade e engenharia de tráfego na arquitetura MPA.

Essa plataforma teria o funcionamento de um *middleware*, pois tem como objetivo atender as necessidades de uma diversidade de aplicações. Os desenvolvedores utilizam interfaces de nível mais alto, o que esconde a complexidade da rede e protocolos de nível mais baixo. Os serviços de *middleware* são definidos por APIs (*Application Programming Interface*) e protocolos que devem suportar, podendo ter diversas implementações que se adequem às especificações da interface e protocolo [5]. Serviços de *middleware* possuem diversas propriedades que os definem: não constituem aplicações ou serviços específicos, mas são genéricos para diversas aplicações; são distribuídos e suportam protocolos e interfaces padronizadas [5].

A plataforma foi proposta como um conjunto de serviços de infra-estrutura para arquiteturas de mobilidade denominado *Mobility Infrastructure Services* (MIS). Na plataforma MIS, os serviços permitem a comunicação com os componentes de modo padronizado, compartilhando funcionalidades, dando suporte e simplificando a construção de novos módulos de gerência de mobilidade, gerência de rede e engenharia de tráfego. O presente trabalho apresenta o desenvolvimento da plataforma MIS com detalhamento de seus componentes.

1.2 Objetivos

Este trabalho tem como objetivo desenvolver uma plataforma que forneça um conjunto de serviços comuns à aplicações de gerência de mobilidade, gerência de redes e engenharia de tráfego em arquiteturas de mobilidade. Os objetivos específicos são:

- Estudo das funcionalidades necessárias e levantamento de requisitos;
- Descrição e especificação dos serviços e arquitetura da plataforma;

- Modelagem da plataforma;
- Implementação da plataforma;
- Testes de validação da plataforma;
- Cenários de utilização da plataforma em aplicações de gerência de mobilidade e engenharia de tráfego.

1.3 Contribuições

No presente trabalho foram conduzidos estudos em arquiteturas de mobilidade, com foco na arquitetura MPA. Ao explorar aplicações em gerência de mobilidade e engenharia de tráfego constatou-se que era necessário um conjunto de serviços para fornecer funcionalidades comuns a essas aplicações.

A contribuição deste trabalho é a proposta e o desenvolvimento de uma plataforma que reúne um conjunto de serviços para prover funcionalidades que sejam comuns a aplicações de gerência de mobilidade, engenharia de tráfego e gerência de rede, focada em arquiteturas de mobilidade.

Essa plataforma atua como uma camada intermediária entre aplicações de um nível mais alto e a arquitetura de mobilidade, padronizando a comunicação entre componentes, acesso aos recursos da rede, armazenando eventos e provendo autenticação. A plataforma proposta facilita o desenvolvimento de aplicações, assim como a integração com outros componentes, como por exemplo, agentes que controlam recursos da rede.

1.4 Organização do texto

O trabalho está organizado da seguinte forma. O Capítulo 2 apresenta uma descrição do contexto de mobilidade e algumas soluções disponíveis. É abordado com um maior nível de detalhes a arquitetura MPA, motivação para o desenvolvimento da plataforma MIS.

O Capítulo 3 apresenta a plataforma MIS e descreve cada um dos serviços suportados. Posteriormente, no Capítulo 4 é conduzida uma modelagem detalhada da plataforma proposta e identificadas as tecnologias empregadas na implementação da mesma.

O Capítulo 5 é dedicado a dois estudos de casos de uso da plataforma MIS. Por fim, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Mobilidade em redes IP

A proliferação e o crescimento do uso de dispositivos móveis com acesso a Internet causa a necessidade de manter a conexão conforme o usuário mude de um local para outro, ou seja de uma subrede para outra. Este movimento ocasiona uma troca de pontos de conexão estabelecendo novos enlaces e alterando configurações de parâmetros de rede. Para o usuário móvel tal movimento deve ser imperceptível quando comparado a utilização de uma máquina estática Durante um percurso devem ser transparentes estas mudanças para as aplicações, sem que haja interferência alguma em seu funcionamento. Esta transparência é um requisito básico, já que é inviável forçar usuários móveis trocarem suas aplicações por outras que sejam compatíveis com a mobilidade [4]. A transparência de migração significa que um usuário não necessita executar nenhum tipo de ação especial, como por exemplo, a reconfiguração manual, antes, durante ou depois da migração do nó móvel [6].

Em redes de comunicação IP como a Internet, ao considerarmos um endereço IP, verifica-se que o mesmo possui dupla funcionalidade. Em uma primeira abordagem, os endereços são utilizados como identificadores únicos, e que deveriam ser mantidos constantes, independente da mobilidade do nó [7]. Neste sentido, os endereços IP estariam sendo equivalentes em funcionalidade com os *Fully Qualified Domain Names* (FQDNs). Ou seja, um endereço IP poderia ser utilizado para identificar um nó dentre milhões de nós que compõe a internet [8]. Contudo, em uma outra abordagem, os endereços IP atuam como ponteiros de localização, e que deveriam mudar conforme um nó móvel se movimenta [7]. Esta função de localização remete a encontrar uma rota entre dois pontos. Tal rota não necessita ser a mesma em ambas as direções, dado que uma rota selecionada para um datagrama depende somente do endereço IP de destino (outros parâmetros como IP de origem, tamanho do *payload*, não interferem nesta rota). Um fator que pode ter relevância na escolha de uma rota é o estado atual da rede. Neste caso, uma rota que poderia ser tipicamente selecionada por um roteador pode ser alterada, caso haja tráfego excessivo, atraso ou descarte devido a congestionamento na rede [8].

Ao analisar estas duas funcionalidades principais de endereços IP é verificado uma situação de contradição quando pensamos em um cenário atual de mobilidade. Um nó móvel necessita ter um endereço IP estático e constante por meio do qual ele é identificado, não afetando as camadas altas de rede. Por outro lado, se este endereço é constante, a rota para este nó móvel consequentemente será estática, os pacotes serão encaminhados sempre para o mesmo local, anulando assim a mobilidade. Como solução, as informações de localização e identificação de um nó móvel devem estar contidas em locais distintos e propagadas através da rede quando requeridas [6]. A proposta mais aceita atualmente é o protocolo *Mobile* IP [2] que estende o protocolo IP permitindo que um nó móvel

efetivamente utilize dois endereços IP, um para identificação e outro para roteamento [8].

Outra proposta para o problema de localização e identificação do nó móvel é o protocolo HIP (*Host Identity Protocol*) [9]. Esse protocolo apresenta como principal característica introduzir um novo espaço de nomes entre as camadas de rede e transporte para identificação do nó móvel [9].

2.1 MIP - *Mobile* IP

O protocolo *Mobile* IP (MIP ou MIPv4) surgiu pela necessidade de um mecanismo escalável que suporte a mobilidade de um nó na Internet. MIP permite que nós mudem seus pontos de conexão na internet sem alterar seus endereços IPs e que um nó móvel seja capaz de manter comunicação com outros nós mesmo após mudar seu ponto de conexão na internet e ainda mantendo seu endereço IP inalterado. MIP também pressupõe que um nó móvel possa se comunicar com outros nós comuns, que não implementem qualquer função de mobilidade. MIP não efetua nenhuma modificação em protocolos de entidades intermediárias do processo de comunicação que não estejam atuando na arquitetura MIP [2].

De maneira geral, MIP permite que um nó móvel possua um endereço estático denominado *Home Address*, utilizado como identificador para camadas de nível mais alto, e outro endereço, chamado *Care of Address* (CoA), que pode ser alterado conforme o nó móvel muda de um ponto de conexão para outro. Este endereço é responsável pela localização atual do nó móvel, permitindo que os pacotes sejam roteados para o novo ponto de conexão. O esquema de funcionamento do MIP pode ser pensado como um trabalho conjunto de três subsistemas. O primeiro é um mecanismo de descoberta. Neste mecanismo, o nó móvel pode determinar seus novos pontos de conexão conforme se movimenta dentro da Internet obtendo um endereço CoA. No segundo subsistema, o nó móvel deve registrar seu endereço CoA em um agente da sua rede de origem. E por último, MIP define mecanismos simples para entrega de pacotes quando o nó móvel se encontra fora de sua rede de origem [8].

No protocolo MIP existe um roteador especial, o *Home Agent* (HA), que pertence à rede de origem do nó móvel e é responsável por tunelar pacotes até o nó móvel quando o mesmo encontra-se em uma rede diferente da sua rede de origem. O roteador HA também deve manter informações sobre a localização atual do nó móvel, como o seu endereço CoA. Há também um outro roteador diferenciado na rede visitada pelo nó móvel, o *Foreign Agent* (FA), que deve fornecer serviços de roteamento para o nó móvel, enquanto ele permanecer conectado nesta rede. O roteador FA desencapsula os datagramas tunelados pelo roteador HA do nó móvel, e também pode atuar como o roteador padrão do nó móvel.

De acordo com a RFC 3344 [2], o funcionamento do protocolo MIP é definido pelos seguintes passos:

- Agentes HA e FA anunciam sua presença na rede através de mensagens *Agent Advertisement*;
- Um nó móvel tem a opção de solicitar tais mensagens através de qualquer agente por meio de um tipo de mensagem chamada *Agent Solicitation*;
- Um nó móvel ao receber mensagens *Agent Advertisement* pode determinar se ele se encontra em sua rede de origem, ou em uma rede visitada;
- Um nó móvel, ao perceber que está localizado em sua rede de origem, pode operar então sem os serviços de mobilidade. Caso esteja retornando a sua rede de origem após ter sido registrado

em uma outra rede, o nó móvel deverá retirar seus registros referentes a mobilidade do roteador HA, através da troca de mensagens do tipo *Registration Request* e *Registration Reply*;

- Em uma situação que um nó móvel detecte que se moveu para uma rede visitada, ele executa procedimentos para obter um endereço CoA nesta rede. Tal endereço pode ser determinado pelo roteador FA ou por algum mecanismo externo como o DHCP (*Dynamic Host Configuration Protocol*);

- No caso de estar operando fora da sua rede de origem e após ter obtido um endereço CoA na rede visitada, o nó móvel deverá registar seu novo endereço CoA em seu roteador HA. Tal procedimento será feito através da troca de mensagens do tipo *Registration Request* e *Registration Reply*, possivelmente, por intermédio do roteador FA;

- Na comunicação do nó móvel com algum outro nó, os datagramas enviados ao seu *Home Address* deverão ser interceptados pelo roteador HA e então tunelados para o endereço CoA referente ao nó móvel. Estes datagramas serão então recebidos na saída do túnel e entregues ao nó móvel;

- Em um sentido oposto, os datagramas que forem enviados do nó móvel para outro nó correspondente serão entregues ao seu destino utilizando mecanismos de roteamento IP padrão, sem ter a necessidade de passar pelo roteador HA.

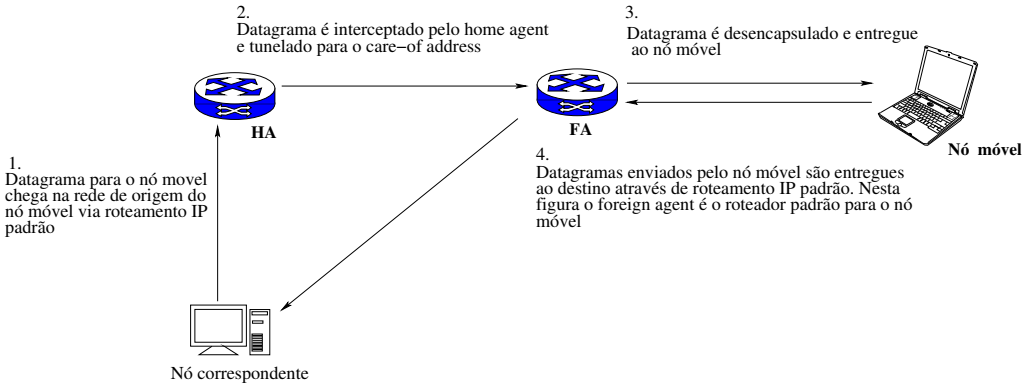

Fig. 2.1: Operação do *Mobile* IPv4.

De acordo com funcionamento do protocolo MIP os datagramas que vão para o nó móvel necessitam passar pelo roteador HA quando o nó móvel se encontra fora de sua rede de origem. Já os datagramas que vão do nó móvel para um nó correspondente seguem o roteamento IP padrão para o seu destino, como pode ser observado na Figura 2.1. Este tipo de roteamento, é chamado roteamento triangular [8] e está longe de ser otimizado. Se os nós em comunicação estão próximos e o roteador HA de um outro lado da Internet, há um consumo desnecessário de banda e recursos em muitos enlaces e roteadores.

2.1.1 MIPv6 - *Mobile* IPv6

O protocolo MIP permite a mobilidade na camada de rede e transparência para as demais camadas. Porém, devido as desvantagens apresentadas, o IETF introduziu o protocolo *Mobile* IPv6 (MIPv6) como um sucessor do *Mobile* IPv4 e com suas deficiências solucionadas [10]. MIPv6 possibilita um roteamento otimizado onde é permitido que um dispositivo móvel se comunique com outro nó utilizando o menor caminho possível provido pelo roteamento IP [11].

MIPv6 compartilha muitas características do MIPv4, beneficiando-se das experiências obtidas no desenvolvimento do protocolo, e propondo soluções para suas deficiências. MIPv6 aproveita-se das novas características fornecidas pelo protocolo IPv6 para implementar estas melhorias. Algumas das melhorias notadas no protocolo MIPv6 de acordo com a RFC 3775 [1] são:

- Não há necessidade dos roteadores FA. MIPv6 é capaz de operar em qualquer rede sem nenhum suporte especial requerido;
- A otimização de rota é parte fundamental e nativa do protocolo, não uma extensão tal como no protocolo MIPv4;
- A otimização de rota do protocolo MIPv6 pode operar seguramente mesmo sem associações de segurança pré definidas, podendo ser empregada em escala global entre todos os nós móveis e seus nós correspondentes;
- Possibilita a coexistência da otimização de rota com roteadores que executam *ingress filtering*[1];
- Os pacotes que são enviados para um nó móvel quando encontra-se em uma rede visitada, são enviados utilizando o cabeçalho de roteamento do IPv6, não necessitando encapsulamento tal como no protocolo MIPv4.

Com estas melhorias os nós móveis não necessitam enviar mensagens de controle de forma separada dos datagramas IP dado que estas informações estão embutidas no próprio datagrama IP. Os datagramas que são enviados de um nó móvel para um correspondente, utilizam o CoA como endereço de origem, e paralelamente o home address é adicionado no datagrama. Sendo assim, a pilha IPv6 do nó correspondente esconderá o endereço CoA, expondo apenas o *Home Address* às camadas superiores.

No modelo de comunicação com otimização de rota do protocolo MIPv6, que é opcional, um nó móvel envia uma mensagem (*binding update*) que contém seu endereço CoA atual para o nó correspondente. Este nó armazena estas informações de localização em um *cache* de *binding*. Este *cache* é consultado quando o nó correspondente envia um datagrama para o nó móvel. Verifica-se neste *cache* se há uma entrada válida para este nó, esta entrada contém uma associação entre os endereços *Home Address* e CoA. Caso exista esta entrada o datagrama será roteado para o endereço CoA do nó móvel diretamente, eliminando um tráfego desnecessário com o roteador HA. No modelo tradicional de comunicação, sem a otimização de rota, não há a necessidade do nó correspondente ter suporte a otimização do protocolo MIPv6. Neste caso, o nó móvel não envia mensagens de *binding update* para ele: os pacotes são roteados do nó correspondente para o roteador HA e então tunelados

[1] É uma técnica especificada pela RFC2827 que verifica nos pacotes ingressantes se o endereço de origem pertence à rede de onde eles estão vindo.

para o nó móvel. No caminho reverso, os pacotes irão por túnel do nó móvel para o roteador HA que então serão encaminhados para o nó correspondente [1].

2.2 HIP - *Host Identity Protocol*

O protocolo HIP propõe a criação de um novo espaço de nomes para identidade de nó (*Host Identity*), que preenche o espaço entre os endereços IP e os nomes identificadores FQDN. Este novo espaço de nomes consiste em identificadores HI (*Host Identifier*), que representam um nome único, global e estático para identificar um nó. O identificador HI possui uma ligação dinâmica com o endereço IP, que atua como localizador do nó sendo responsável por desempenhar funções de roteamento. Este identificador é codificado e constituído por um par de chaves assimétricas. Cada nó possui pelo menos um identificador HI, porém pode possuir mais de um. A diferença entre identidade e identificador HI é que o identificador se refere a uma entidade abstrata que pode ser identificada, já o identificador se refere a um padrão concreto que é usado no processo de identificação [9].

O identificador HI é criado por um nó HIP a partir de um valor aleatório e tem um significado global no sistema de nomes. O identificador HI é uma chave formada por um par de valores público-privado, sendo que o valor público é acessível por todos os nós HIP e o privado representa a identidade do nó, sendo que somente ele a possui. A camada de identidade do nó pode criar uma marca HIT (*Host Identity Tag*) que é uma representação de 128 bit para uma identidade de nó. O valor da marca HIT é calculado por uma função *hash* sobre o correspondente identificador HI [9]. As vantagens de se usar um *hashing* é que o tamanho é fixo, facilitando a utilização por protocolos, e também não há a necessidade de uma entidade externa para identificação.

Na arquitetura HIP, o identificador e localizador de um nó são separados, de forma que o endereço IP continua atuando como localizador sendo responsável pelo roteamento e o identificador HI atua como o identificador do nó. O protocolo HIP atua entre as camadas de rede e transporte permitindo o desacoplamento das camadas superiores com os endereços IP. A identificação do nó é feita na camada de identidade introduzida pelo protocolo HIP. As diferenças entre as ligações das entidades lógicas na arquitetura tradicional e HIP podem ser observadas na Figura 2.2. Pode-se verificar que o espaço de nome identidade é responsável pela identificação do nó na arquitetura HIP enquanto na arquitetura tradicional o endereço IP seria o responsável por tal função.

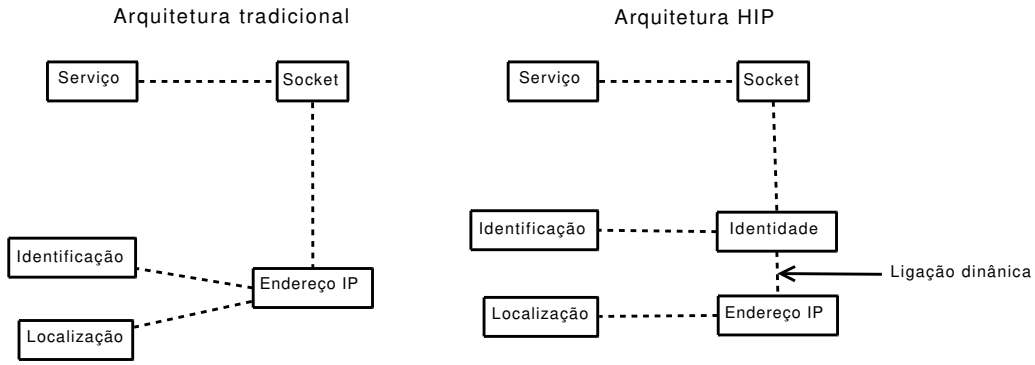

Fig. 2.2: Ligações de entidades lógicas da arquitetura HIP e tradicional.

HIP provê mobilidade no sentido que as associações da camada de transporte estão ligadas as

identidades HI, desta forma não são interferidas as conexões de transporte caso o endereço IP do nó móvel mude. Quando um nó efetua um handover durante a comunicação, seu endereço IP muda, e, desta forma, o nó móvel deve enviar um pacote *readdress* HIP para informar ao nó correspondente seu novo endereço IP. O nó correspondente deve efetuar uma atualização para que o nó móvel seja atingido através deste novo endereço IP. Neste processo de atualização pode ser utilizado um mecanismo chamado *rendezvous*. Este mecanismo armazena as informações do identificador HI de cada nó móvel associando ao seu endereço IP atual. O nó correspondente pode acessar este mecanismo obtendo associações atualizadas do endereço IP com o identificado HI do nó móvel.

2.3 Soluções de micromobilidade

MIPv6 e MIPv4 apresentam solução para mobilidade de maneira global, ou macromobilidade, que se refere à mobilidade através de diversas redes e domínios. MIP supõe que o movimento de nós móveis entre pontos de acesso (AP) pertencentes a uma mesma subrede é gerenciado pela camada 2 do protocolo sem envolver mecanismos de camada 3. Porém, se um nó móvel se conecta em um AP em uma outra subrede do mesmo domínio, o endereço IP do nó móvel não é mais topologicamente válido. Sendo assim, há a necessidade de atualização de localização, ou *binding updade* do protocolo MIP, o que gera um tráfego alto de sinalização e uma latência considerável durante o processo de troca de pontos de acesso (*handover*). Esta mobilidade dentro de um único domínio de rede, é chamada micromobilidade. O protocolo MIP utiliza o mesmo mecanismo para ambos os casos, macro e micromobilidade [10] [12] [3].

Os problemas se intensificam quando um nó móvel efetua uma alta taxa de *handover* em uma área de cobertura de pequenas celulas, provocando com isto a necessidade de *binding update*, para todo *handover*. Como os *overheads* de sinalização altos impostos pelo protocolo MIP aumentam a latência e a perda de pacotes durante um *handover*, haverá severa interferência nas aplicações dos usuários. Como solução a este problema, têm sido propostos diversos protocolos de micromobilidade [13] [14] que apresentam como objetivo melhorar a mobilidade local, propiciando *handovers* rápidos, diminuindo *overheads* e perda de pacotes, de maneira a causar menor interferência para as aplicações dos usuários.

2.3.1 FMIPv6 - *Fast Handovers for Mobile* IPv6

Fast Handovers for Mobile IPv6 (FMIPv6) [15] é uma extensão do protocolo MIPv6 que tem como objetivo reduzir a latência e a perda de pacotes durante um *handover*. No protocolo FMIPv6 quando um nó móvel detecta que vai se mover para uma nova subrede, o mesmo pode adquirir informações para obter um endereço CoA para o novo ponto de conexão enquanto ainda está conectado em sua rede atual. Isto também permite que o nó móvel use este novo endereço CoA imediatamente quando estiver conectado [16]. Um nó móvel pode descobrir possíveis pontos de acesso aos quais pode se conectar, utilizando mecanismos de camada de enlace específicos, como um *scan* em uma WLAN, e então requerer informações da subrede correspondente para um ou mais destes pontos de acesso descobertos [15].

FMIPv6 procura eliminar a latência que é envolvida durante um processo de atualização de localização, ou *binding update*. Para isto, um túnel bidirecional do roteador de acesso anterior para o

novo é estabelecido, e paralelamente os procedimentos de *binding update* estão sendo executados.

O procedimento que é executado pelo protocolo FMIPv6 quando um nó móvel está para efetuar um *handover* é expresso pelas seguintes etapas [17]:

- O nó móvel deve obter um novo endereço CoA com o novo roteador de acesso enquanto ainda está conectado ao anterior;
- O nó móvel deve enviar uma mensagem de *binding update* para o seu roteador de acesso, para que este, atualize seu *cache* para o novo endereço CoA do nó móvel;
- O roteador de acesso atual deve começar a redirecionar pacotes destinados ao nó móvel para o novo roteador de acesso.

Um ponto relevante no protocolo FMIPv6 é que os roteadores de acesso necessitam ter conhecimento dos roteadores de acesso que estão localizados próximos onde um nó móvel deseja se conectar. Para isto é necessário conhecimento dos pontos de acesso associados a cada roteador de acesso. Tais requisitos sugerem que uma implementação do protocolo é facilitada em redes intra-organizacionais [16].

Comparando-se FMIPv6 com a operação convencional do protocolo MIPv6, verifica-se vantagens no protocolo FMIPv6 que procura ser mais eficiente eliminando o atraso introduzido pela configuração de endereço e os procedimentos de *binding update* que são executados pelo nó móvel junto ao roteador HA e o nó correspondente [16]. Isto porém é atingido com algum custo, já que existe um aumento no número de mensagens de sinalização, entre o roteador de acesso e o nó móvel, visando prover um procedimento de *handover* antecipado [17]. FMIPv6 também requer que a implementação do protocolo MIPv6 em nós móveis seja atualizada, para que incorpore mobilidade local e *handover* antecipado [12].

2.3.2 HMIPv6 - *Hierarchical Mobile* IPv6

Hierarchical Mobile IPv6 (HMIPv6) introduz um novo elemento, o *Mobility Anchor Point* (MAP). MAP é uma entidade local para auxiliar o *handover* e pode ser localizado em qualquer nível da hierarquia da rede, incluindo um roteador de acesso. O objetivo do roteador MAP é limitar a quantidade de sinalização interdomínio e com isto restringir localmente as sinalizações de *handover*. Um nó móvel envia mensagens de *binding update* para o respectivo roteador MAP local, evitando que tal sinalização seja direcionada ao roteador HA fora do domínio, que usualmente está em uma rede distante [18].

Quando um nó móvel chega em um domínio de uma rede visitada, há a descoberta dos roteadores MAP existentes, e informações de seus endereços, e respectivas distâncias, através de mensagens de *Router Advertisement*. O nó móvel configura dois endereços CoA, um referente a sua localização dentro do domínio local (LCoA), sendo um endereço roteável que tem o mesmo prefixo do roteador padrão do nó móvel, e outro referente a subrede do roteador MAP, indicando um endereço regional (RCoA).

Em seguida o nó móvel se registra com o seu roteador HA enviando o seu endereço RCoA, que é associado ao seu endereço *home address*. Um registro local no roteador MAP também é necessário, onde o nó envia o seu endereço LCoA que é associado ao endereço RCoA no *cache* de *binding* do roteador MAP. Sendo assim, MAP atua como um roteador HA local que irá receber todos os pacotes

pertencentes ao nó móvel que está servindo, encapsulá-los e tunelá-los diretamente para o endereço atual (LCoA) do nó móvel [19].

Ao se mover dentro do mesmo domínio do roteador MAP, o nó móvel muda apenas o seu endereço LCoA, e portanto só necessitará se registrar junto ao roteador MAP. O endereço RCoA não se modificará enquanto o nó se movimenta dentro do domínio, tornando a mobilidade transparente para o roteador HA e os nós correspondentes [18]. Deste modo, a operação do roteador HA e dos nós correspondentes não é afetada, porém os nós móveis necessitam de uma atualização na implementação do protocolo MIPv6 para que suporte ao protocolo HMIPv6 [12].

2.3.3 PMIPv6 - *Proxy Mobile* IPv6

Usando uma abordagem diferente do protocolo MIP, o *Proxy Mobile* IPv6 (PMIPv6) foi proposto para prover suporte ao gerenciamento de mobilidade IP centrada na rede, sem necessitar da participação do nó móvel em qualquer sinalização de mobilidade. Para tanto, entidades de mobilidade na rede cuidarão do suporte à movimentação do nó iniciando sinalizações necessárias [20].

As entidades funcionais principais do protocolo PMIPv6 são o *Mobile Access Gateway* (MAG) e o *Local Mobility Anchor* (LMA). Quando um nó entra pela primeira vez no domínio e se associa a um ponto de acesso, ele deve ser autenticado. O roteador MAG determina se as políticas de autenticação foram satisfeitas e, em seguida, o roteador MAG estabelece um túnel bidirecional com o roteador LMA. Este túnel é compartilhado por todos os nós móveis conectados a partir deste roteador MAG ao roteador LMA e possibilita o nó móvel utilizar um endereço com prefixo de sua rede de origem. MAG é responsável por detectar os movimentos do nó móvel e iniciar sinalizações relacionadas a mobilidade com o roteador LMA.

Da perspectiva de um nó móvel, o domínio todo se apresenta como sua rede de origem. Sendo assim, é desnecessário configurar um endereço CoA no nó móvel. Quando um nó móvel se movimenta para longe do roteador MAG que o está servindo, o túnel não é removido, pois ainda é utilizado por outros nós. Este túnel só será removido quando o seu tempo de vida expirar ou quando não houver nenhum nó móvel associado a ele [21].

O roteador LMA é similar a um roteador HA no MIPv6, apresentando algumas capacidades adicionais que são necessárias para o suporte do PMIPv6. O papel principal do roteador LMA é manter alcançáveis os endereços do nó móvel enquanto ele se move dentro de um domínio PMIPv6. LMA inclui uma entrada no *cache* de *binding* para cada nó móvel atualmente registrado. Esta entrada contém informações adicionais referentes ao nó móvel, que o associa com seu roteador MAG servidor, possibilitando o relacionamento entre os roteadores MAG e LMA [22].

Conforme um nó móvel se move dentro do domínio, os túneis entre MAG e LMA, e as rotas no roteador LMA são atualizadas. Desta forma, quando o roteador LMA recebe um pacote endereçado para o nó móvel ele o encaminha para o túnel referente ao roteador MAG que o nó está associado [12].

2.3.4 MPA - *Mobility Plane Architecture*

MPA é uma arquitetura de micromobilidade em redes IP, MPLS/GMPLS. O objetivo desta arquitetura é tornar o processo de *handover* mais rápido para minimizar a interrupção na comunicação quando um nó móvel mudar seu ponto de conexão. Um dos objetivos desta arquitetura é colocar os

requisitos necessários para micromobilidade, na rede, e não nos nós móveis. O ponto chave da arquitetura MPA é empregar uma rede sobreposta sobre uma rede de transporte para direcionamento de tráfego aos nós móveis. Esta rede sobreposta emprega túneis ponto-multiponto (P2MP) para encapsular o tráfego aos nós móveis. Um outro requisito desta arquitetura é idealmente utilizar, protocolos de rede bem estabelecidos [3] [12].

A rede sobreposta é composta por elementos chamados *Mobility Aware Router* (MAR), que são roteadores providos de funcionalidades da arquitetura MPA. A arquitetura funcional e os elementos básicos que compõem a arquitetura MPA podem ser observados na Figura 2.3.

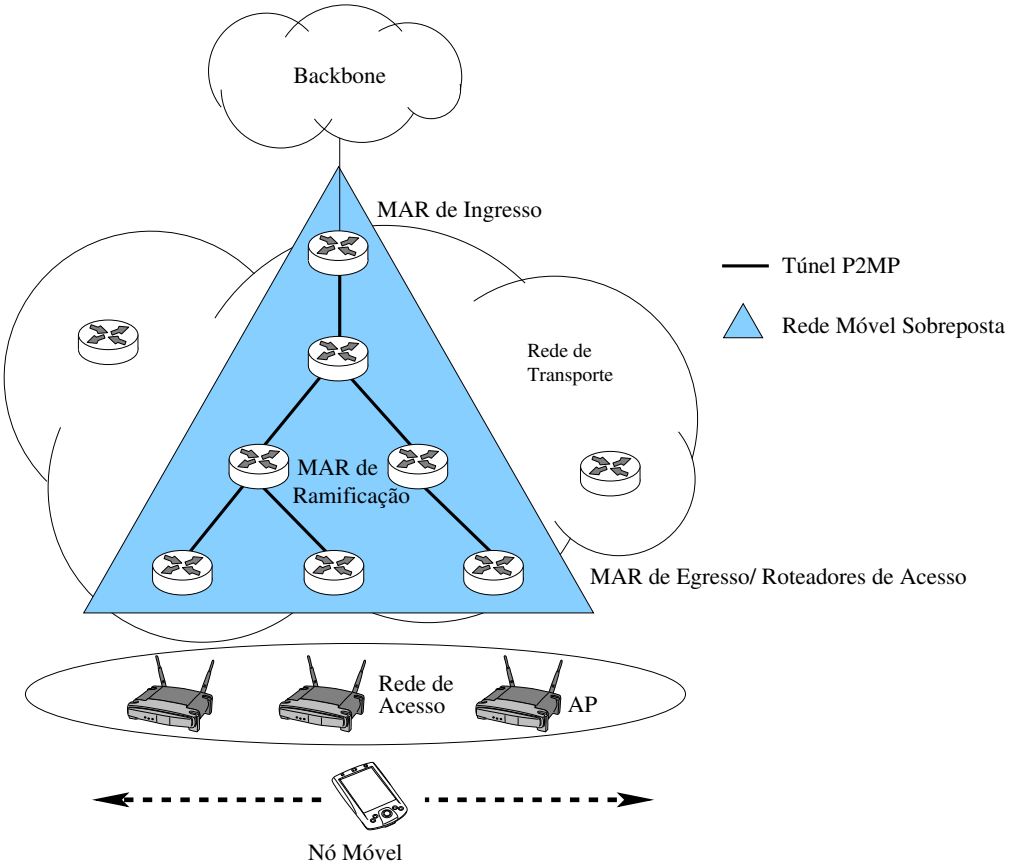

Fig. 2.3: Arquitetura funcional e os elementos básicos da MPA.

A rede de transporte, é uma rede IP, MPLS/GMPLS na qual se deseja oferecer os serviços de micromobilidade. MAR é um roteador que incorpora funções relacionadas a mobilidade na rede de trasporte, sendo responsável pelo tunelamento e redirecionamento de tráfego para os nós móveis. Os túneis P2MP formam uma topologia de árvore, sendo que os nós são roteadores MAR e os arcos são segmentos de túneis. A raiz da topologia é um roteador MAR de ingresso, as folhas são roteadores MAR de egresso, além de roteadores MAR de ramificação que se localizam nos níveis intermediários desta topologia. A rede lógica é constituída pelos túneis, que são responsáveis pelo encaminhamento de tráfego para os nós móveis.

Segundo [3] a operação básica da arquitetura MPA é definida por meio de quatro blocos funcionais:

- Gerenciamento de Túneis (GT): Este bloco funcional é responsável pelo estabelecimento, desativação e re-roteamento de túneis P2MP. Para tal, é necessário que este disponibilize interfaces de interação com o sistema de gerenciamento de rede e também com operadores humanos. Gerenciamento de túneis é uma atividade desempenhada pelos roteadores MAR;
- Roteamento Móvel (RM): RM rastreia a posição atual do nó móvel (respectivo roteador de acesso) e atua sobre os roteadores MAR de maneira que o tráfego seja redirecionado para a nova posição do nó móvel. Esta função é desempenhada exclusivamente pelos roteadores MAR;
- Configuração de Endereço (CE): Esta entidade é responsável por configurar um endereço de camada 3 para o nó móvel no momento que ele se conecta a rede ou quando muda de rede de acesso. A configuração de endereço é um procedimento no qual tanto a rede (roteadores MAR) quanto o nó móvel cooperam;
- Facilitador de Handover (FH): Este bloco funcional tem como objetivo prover serviços que facilitem o processo de *handover* do nó móvel dentro da mesma rede de acesso. Isso inclui notificação de associação ao roteador MAR de egresso. Este bloco funcional envolve MARs, pontos de acesso, servidores de autenticação e o próprio nó móvel.

Quando um pacote endereçado para um nó móvel chega a um roteador MAR de ingresso, ele é tunelado até chegar ao roteador MAR de egresso que é capaz de rotear o pacote para o nó móvel. Quando um nó móvel executa um *handover*, o bloco CE no nó móvel e na rede interagem para prover um endereço de camada 3 para o nó móvel. Se o endereço é idêntico ao anterior as conexões de transporte não são quebradas devido ao *handover*. O processo de *handover* também gera uma atualização de localização para o bloco RM. O processo de atualização de localização atualiza a tabela de roteamento móvel nos roteadores MAR, de tal forma que um pacote endereçado ao nó móvel é roteado para o roteador MAR de egresso referente ao enlace que o nó está conectado.

As atualizações nas tabelas de roteamento móvel são feitas assim que as mensagens do protocolo de roteamento móvel indicando o novo ponto de conexão são processadas pelos roteadores MAR. As entradas na tabela são *soft-state*, significando que as entradas são descartadas se mensagens de atualização de localização cessam (param de chegar). *Soft-state* é uma maneira de descartar rotas de nós móveis quando eles não são mais atingidos por estas rotas. Este esquema requer que um nó móvel confirme sua presença na rede periodicamente para que as rotas sejam atualizadas. Uma maneira de forçar um nó móvel executar esta ação é prover um endereço de camada 3 com um tempo de validade curto, forçando o nó móvel executar renovações de endereço periódicas que irão disparar mensagens de atualização de localização [3]. Uma maneira mais eficiente é o ponto de acesso notificar a conexão e desconexão de nós móveis.

Gerência de Mobilidade e Engenharia de Tráfego na MPA

Na arquitetura MPA, visando melhorar o desempenho da comunicação do nó móvel durante o *handover*, foram definidos componentes responsáveis por prever e por antecipar as ações necessárias no caso de *handover* [4]. O objetivo é tornar a interrupção da comunicação causada pelo *handover* mais suave, com menor perda de pacotes deixando transparente para camadas superiores a ocorrência do *handover*.

Um serviço de previsão do *handover* [4] foi proposto. Este serviço envolve um conjunto de tarefas que são responsáveis por restabelecer a comunicação quando um nó móvel muda de um ponto de acesso para outro, a fim de tornar o *handover* proativo. Para isto, algum tipo de sinalização pode ser trocado entre os roteadores de acesso antes do nó móvel efetuar o *handover*. As informações necessárias para previsão e antecipação do *handover* podem ser obtidas através de um histórico de conexões do nó móvel, obtendo-se padrões de movimento em determinado período de tempo, traçando um perfil de movimento do nó móvel. Este histórico pode ser obtido através do *logging* de eventos de conexão.

Por meio de um perfil de movimentação é possível antecipar a troca de informações necessárias para a conexão do nó móvel ao novo roteador de acesso e também, forçar *bicasting* de pacotes, ou seja, replicar durante o período de *handover* pacotes endereçados ao nó móvel direcionando-os tanto ao ponto de acesso anterior como ao novo ponto de acesso. Assim, o recebimento de pacotes, caso a previsão se efetive, será imediato no novo ponto de acesso.

A engenharia de tráfego é definida como os aspectos da rede que lidam com questões de avaliação e otimização de desempenho [23]. Geralmente inclui a aplicação de princípios tecnológicos e científicos para medir, caracterizar, modelar e controlar o tráfego da rede visando atingir objetivos específicos de desempenho. Os mecanismos para controle de congestionamento de engenharia de tráfego podem ser caracterizados como preventivos ou reativos [24]. Na arquitetura MPA, o controle preventivo é feito mediante a aplicação de algoritmos para a escolha adequada da topologia lógica, isto é, quais roteadores fazem parte da rede sobreposta e por quais roteadores cada segmento de túnel deve passar. Os algoritmos de controle preventivo ainda estão em fase de estudo. O controle reativo é realizado por meio de agentes inteligentes que, em caso de congestionamento ou eminência deste, fazem a reconfiguração de túneis e dos parâmetros das classes de serviço (CoS - *Class of Service*), isto é, os parâmetros das filas de saída dos roteadores são alterados para acomodar o congestionamento do tráfego.

Na arquitetura MPA foi introduzida a diferenciação de tráfego usando-se classes de serviço como definida pela arquitetura DiffServ (*Differentiated Services Architecture*) [25]. Essa implementação utiliza o esquema de controle de tráfego do Linux [26] para estabelecer classes de serviço para o tráfego dos túneis. Em cada interface de rede de cada roteador MAR usando a disciplina *Hierarchical Token Bucket*[2] são criadas três filas de saída. A fila de encaminhamento de prioridade mais alta recebe uma garantia de banda mínima como uma porcentagem da banda disponível para o túnel, por exemplo 60%, a de prioridade média uma quantidade menor, por exemplo 30%, e a de prioridade mais baixa recebe o restante da banda. Estas filas definem três classes de serviço: ouro, prata e bronze. Assim, pode-se, por exemplo, atribuir a classe ouro para o tráfego de vídeo e áudio e a classe prata para o tráfego Web. A classe bronze seria atribuída aos clientes que não estivessem inscritos no serviço com QoS (*Quality of Service*). As filas são removidas quando o túnel ponto-multiponto é destruído. Um sistema de controle de congestionamento recalcula os parâmetros das filas de encaminhamento dos roteadores envolvidos e solicita a reconfiguração desses parâmetros.

A atuação na configuração e reconfiguração dos túneis e das filas de saída dos roteadores a fim de prover um melhor desempenho na rede requer a atuação nos recursos existentes (como os roteadores MAR). Isto nos remete a necessidade de um sistema que seja responsável pela atuação e monitoramento dos recursos, permitindo, respectivamente, ler e modificar o estado dos mesmos.

[2]http://luxik.cdi.cz/ devik/qos/htb/ .

Durante o estudo e implementação de aplicações de gerência de rede, engenharia de tráfego e gerência de mobilidade ficou evidente que estas aplicações demandam um conjunto de serviços comuns como, por exemplo, serviços de *logging*, notificação, acesso centralizado a recursos e autenticação. Este serviços foram agrupados em uma plataforma única descrita no Capítulo 3.

Capítulo 3

Serviços de suporte à mobilidade

Aplicações de gerência de rede, engenharia de tráfego e gerência de mobilidade são importantes quando se deseja obter melhorias no desempenho de *handover* e de tráfego em redes móveis. Estudos de aplicações nesta área foram conduzidos no escopo da arquitetura de micromobilidade MPA e observou-se que tais aplicações apresentavam necessidades comuns, dentre elas:

- Comunicação padronizada entre aplicações;
- Notificação de eventos;
- Armazenamento de eventos para uma análise futura;
- Monitoramento e atuação em recursos ou serviços da rede.

Neste sentido, como solução a estas necessidades verificadas, foi proposto o desenvolvimento de uma plataforma que contemple serviços para suprir estas necessidades, formando uma camada que desse suporte ao desenvolvimento destas aplicações. Esta plataforma, denominada MIS (*Mobility Infrastructure Services*) [27] corresponde ao conjunto de serviços que oferecem as funcionalidades básicas comuns às aplicações de gerência de rede, engenharia de tráfego e gerência de mobilidade. Os serviços providos pela plataforma MIS são:

- Notificação;
- Persistência (*logging*);
- Relatório;
- AAA (*Authentication, Authorization, Accounting*);
- Redirecionamento (*Proxy*).

3.1 Arquitetura MIS

A comunicação com os serviços providos pela plataforma MIS é parte fundamental, já que clientes podem estar na mesma rede ou não. Considerando este aspecto, a plataforma deve possuir acessibilidade independente de onde esteja o cliente, sem requerer a utilização de qualquer software ou protocolo específico. As informações trocadas entre os clientes e a plataforma MIS também necessitam possuir um padrão de fácil manipulação e processamento, empregando um formato bem estabelecido, para que as aplicações clientes possam interagir de forma fácil.

O modelo de comunicação da plataforma é baseado no protocolo HTTP (*Hypertext Transfer Protocol*) [28], por se tratar de um protocolo estabelecido no âmbito da Internet e que trafega na maioria dos sistemas de comunicação de dados sem qualquer tipo e bloqueio. Os dados a serem transportados são documentos no formato XML [29], por se tratar de um formato universal para dados estruturados, escalável e de fácil processamento por aplicações clientes.

A plataforma MIS é estruturada na forma de componentes que são acessados pelos clientes através de suas respectivas URIs (*Universal Resource Identifier*). A Figura 3.1 apresenta a arquitetura de componentes da plataforma MIS e suas interações com as aplicações cliente de nível mais alto e com aplicações ou agentes de um nível mais baixo. Os agentes neste trabalho correspondem a componentes de software tais como agentes que controlam recursos e utilizam o protocolo SNMP (*Simple Network Management Protocol*) [30].

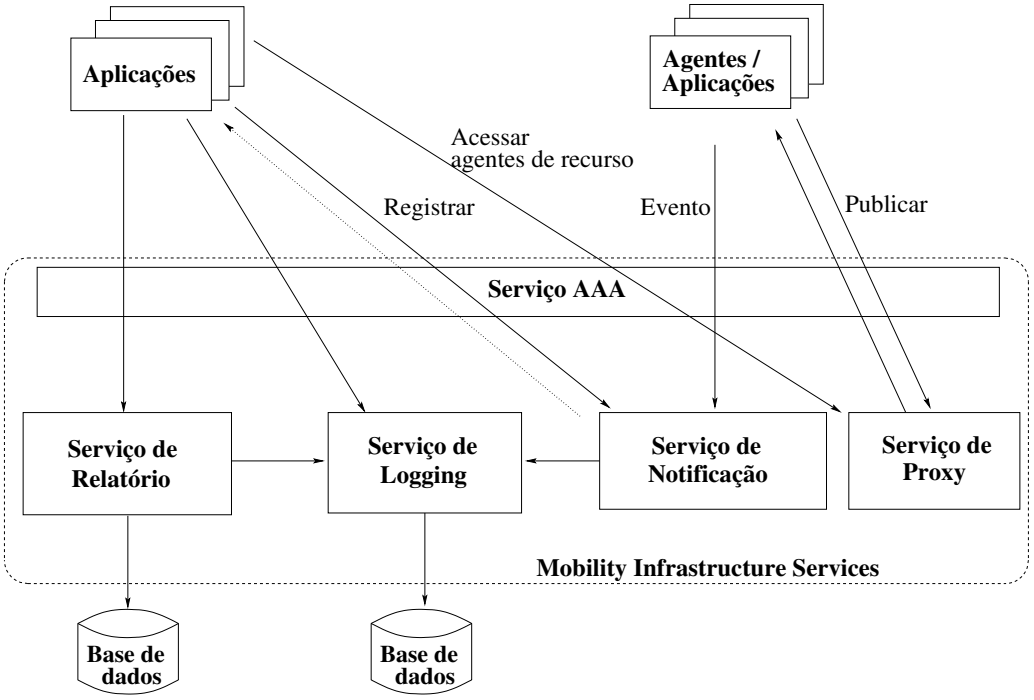

Fig. 3.1: Plataforma Mobility Infrastructure Services.

3.2 Serviço de Notificação

Um sistema de comunicação entre as diversas aplicações e agentes presentes na rede é parte essencial, pois é através dele que as informações trafegam de um ponto ao outro e são difundidas por toda a rede. Para tanto foi proposto o serviço de notificação, que tem como objetivo permitir uma forma padronizada e centralizada para comunicação entre aplicações.

O serviço de notificação será o ponto único para comunicação, por onde toda informação de notificação passará, facilitando o envio e recebimento de mensagens (eventos). Este serviço também é importante para a persistência, pois terá comunicação com o serviço de *logging*, detalhado na Seção 3.3. A interação com o serviço de notificação será feita utilizando o protocolo HTTP e XML como conteúdo, conforme mencionado na Seção 3.1.

As mensagens do modelo de notificação de eventos são estruturadas em duas partes principais, o cabeçalho (*header*) e o corpo (*body*). O cabeçalho deve conter informações que identificam o evento, devendo possuir uma marca obrigatória, "type", que indica o tipo do evento e pode conter algumas marcas adicionais referentes ao evento que são: "from", que indica quem o enviou; "to", qual é o destino; "ttl", que indica o tempo de permanência em memória no serviço; "log" que indica se o evento é para ser enviado ao serviço de *logging*.

O corpo se refere ao conteúdo do evento, podendo possuir qualquer tipo de informação em qualquer estrutura XML, não passando por nenhuma regra de validação que seja restritiva. As mensagens deverão seguir obrigatoriamente este modelo, para que sejam aceitas no serviço de notificação, caso contrário serão rejeitadas gerando um erro ao cliente.

A Figura 3.2 é um exemplo de evento do serviço de notificação. Podem ser identificadas as duas partes principais do evento, o cabeçalho (marca XML "header") e o corpo (marca XML "body").

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <message>
3    <header>
4      <type>TriggerEvent</type>
5      <from>00:0c:42:1b:a7:df</from>
6      <ttl>10000</ttl>
7      <log>true</log>
8    </header>
9    <body>
10     <mn>
11       <mac>00:16:6f:2f:e9:b8</mac>
12       <ip>10.20.9.100</ip>
13       <router>10.20.9.13</router>
14     </mn>
15     <ap>
16       <ip>10.20.9.31</ip>
17       <mac>00:0c:42:1b:a7:df</mac>
18     </ap>
19   </body>
20 </message>
```

Fig. 3.2: Evento XML para *logging*.

Dois modelos de funcionamento foram propostos para o serviço de notificação com objetivo de atender as necessidades de aplicações clientes, sendo eles: o modelos *push* e o *pull*.

3.2.1 Modelo de notificação *push*

O modelo *push* funciona de forma síncrona, ou seja, o cliente receberá notificação quando algum produtor de eventos enviar uma mensagem que seja de interesse do cliente. Neste sentido, o cliente deverá se registrar (ou subscrever) no serviço, passando o endereço onde irá receber as notificações; um filtro pelo qual será verificado se um evento é de seu interesse; e também um tempo de permanência no recebimento de notificações.

O funcionamento do modelo *push* pode ser observado na Figura 3.3. Quando um evento chega no serviço de notificação, é percorrida uma lista que contém todos os clientes que foram registrados para receber notificação, se o filtro passado pelo cliente casa com o evento que chegou, o cliente é notificado.

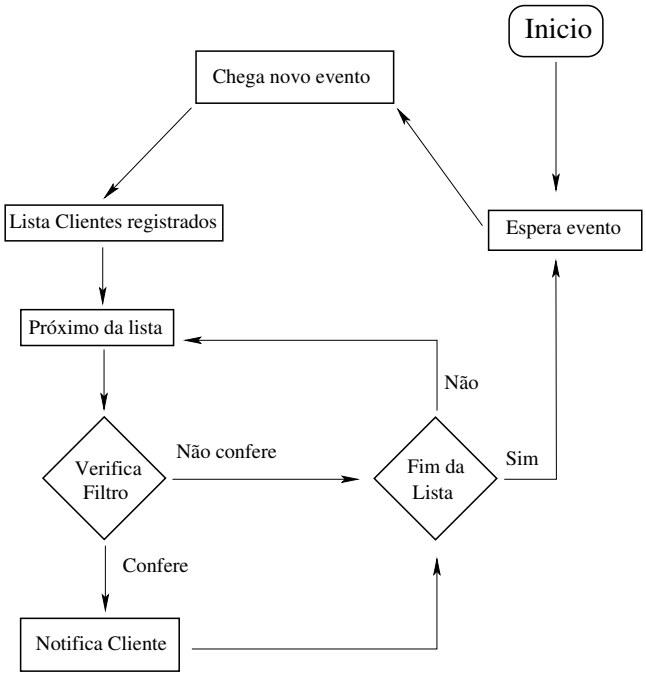

Fig. 3.3: Fluxograma modelo *push* do Serviço de Notificação.

O registro no modelo *push* do serviço é feito através de uma requisição HTTP POST, fornecendo como parâmetros: URI, XPATH e TTL. URI corresponde a um endereço para envio de mensagens contendo o evento; XPATH representa o filtro que é aplicado para determinar qual evento deve ser notificado no formato XPath [31]; e TTL é o tempo de validade do registro, ou seja, o intervalo de tempo durante o qual as notificações vão ser enviadas. O envio das notificações para o cliente também será feito por HTTP POST contendo o evento no formato XML.

3.2.2 Modelo de notificação *pull*

O modelo *pull* tem um funcionamento de forma assíncrona, neste caso, o cliente deve fazer uma requisição ao serviço de notificação, recebendo uma lista de eventos como resposta. Na requisição deve ser passado apenas um filtro XPath, que será responsável por selecionar as mensagens presentes no serviço que sejam de interesse do cliente.

Na Figura 3.4 pode ser observado o funcionamento do modelo *pull*. Quando um cliente faz uma requisição o serviço efetua uma busca nos eventos disponíveis, que confiram com o filtro passado pelo cliente. Caso o resultado da busca seja diferente de vazio, ou seja, caso a busca retorne eventos de interesse do cliente, é montado uma resposta no formado XML com a lista de eventos. Caso contrário, é gerado uma resposta para o cliente indicando que não foram encontrados eventos de seu interesse.

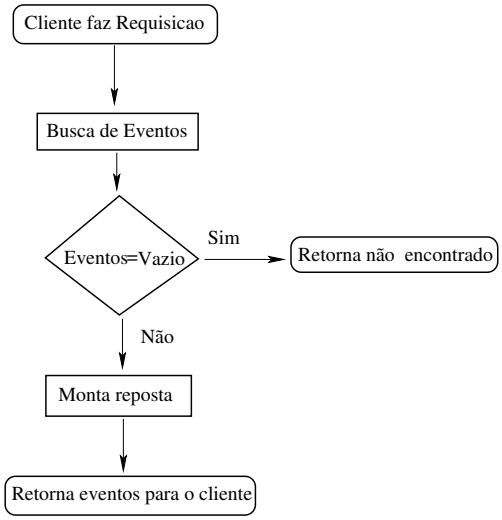

Fig. 3.4: Fluxograma modelo *pull* do Serviço de Notificação.

A requisição do cliente no modelo *pull* é feita através de uma requisição HTTP GET passando como parâmetro um filtro no formato XPath [31]. O serviço, então, retorna uma mensagem XML com uma lista de eventos que estão de acordo com o filtro.

3.3 Serviço de *Logging*

A necessidade do armazenamento de eventos é importante para diversas aplicações, visto que informações registradas podem ser analisadas a posteriori, fato este que motivou o desenvolvimento do serviço de *logging*. Este serviço, é responsável por armazenar de forma persistente eventos que ocorram na rede e sejam difundidos pelo serviço de notificação. Tais eventos podem ser enviados por qualquer aplicação que tenha acesso à plataforma MIS.

A consulta aos dados armazenados deve ser flexível podendo ser efetuada conforme as necessidades das aplicações. Dessa forma, o serviço de *logging* também provê uma interface para consulta de eventos. Uma aplicação poderá acessar esta interface e obter dados fornecendo parâmetros de busca de eventos, por exemplo, tipo de evento, intervalo de tempo, etc.

O serviço deve ter suporte para incorporar novos eventos, não sendo exclusivo de nenhum padrão de conteúdo, desde que o modelo de mensagens XML seja mantido. Desta maneira, conforme novas aplicações necessitem de novos tipos de eventos, estes, poderão ser armazenados e consultados sem que haja mudanças no serviço de *logging*.

Para que um evento possa ser armazenado no serviço de *logging*, quando a notificação contendo as informações deste evento é enviada, sua mensagem deve possuir uma marca “log“, presente no cabeçalho, com o valor marcado como *true*. Esta marca é opcional e, caso não exista, o evento não será armazenado pelo serviço de *logging*. A Figura 3.5 ilustra o funcionamento do serviço e o fluxo de dados do envio de eventos para logging.

Um exemplo de evento enviado para o serviço de *logging* pode ser observado na Figura3.2 com indicação para armazenamento persistente pelo serviço de *logging*. Observe que a marca “log“ na linha 7 do cabeçalho foi definida como *true*. Este evento trata-se de uma notificação de conexão de um nó móvel, onde pode ser notado no corpo do evento a identificação do nó móvel através da marca “mn“ e do referente ponto de acesso que o nó móvel se conectou, com a marca “ap“.

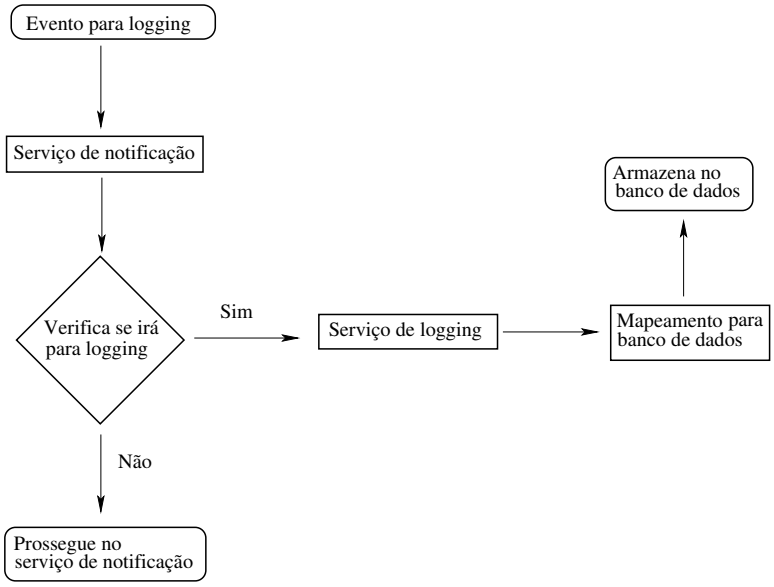

Fig. 3.5: Fluxograma do envio de eventos *logging*

As informações providas por este serviço serão utilizadas por aplicações que necessitem de informações de eventos ao longo do tempo, mostrando-se importante pois através da análise destes dados será possível encontrar padrões de mobilidade, predição de movimentos de nós móveis, antecipar ações de *handover*, detectar e prever falhas, etc.

3.3.1 Armazenamento de eventos

O armazenamento persistente de eventos no serviço é feito por meio de banco de dados. Foram realizados estudos para determinar o tipo de banco de dados apropriado para o serviço de *logging*. Para isto foram analisados dois modelos: XML nativo e objeto-relacional.

Em banco de dados XML nativo, os documentos XML são armazenados em estruturas de dados otimizadas para estes documentos e as consultas empregam linguagem XPath [31] ou XQuery [32]. Já em banco de dados relacional os documentos XML são armazenados em tabelas usando linguagem SQL (*Structured Query Language*) que não é adequada à recuperação de documentos XML.

Para análise de desempenho e flexibilidade foram conduzidos alguns testes que abordam os dois modelos em questão, utilizando-se bancos bem estabelecidos no mercado. Foi utilizado o banco XML nativo de código aberto Sedna[1] e o banco de dados objeto-relacional de código aberto PostgreSQL[2].

No banco de dados Sedna, o armazenamento dos eventos é feito na forma XML sem que seja necessário nenhuma alteração. Entretanto, no PostgreSQL é necessário que seja feito um mapeamento em tabelas. No modelo em questão foram empregadas múltiplas tabelas, baseadas em data, tipo de evento, endereço MAC, endereço IP e marcas presentes no evento (somente o nome da marca). Estas tabelas foram escolhidas visando um melhor desempenho nas consultas, e mantendo um formato genérico nos eventos sem que haja restrições caso um novo evento seja introduzido. O modelo do banco proposto com os relacionamentos entre tabelas pode ser observado na Figura 3.6. De maneira geral existe a tabela "Event" que representa um evento, armazenando o conteúdo e informações temporais. Esta tabela se relaciona com possíveis valores de IP e MAC, que estão presentes nas referentes tabelas. A tabela "Event" também se relaciona com "EventType" que representa o tipo de evento, e esta por sua vez possui diversas marcas expressas na tabela "Tag". Por exemplo, o evento da Figura 3.2 seria mapeado nas tabelas ilustradas na Figura 3.6.

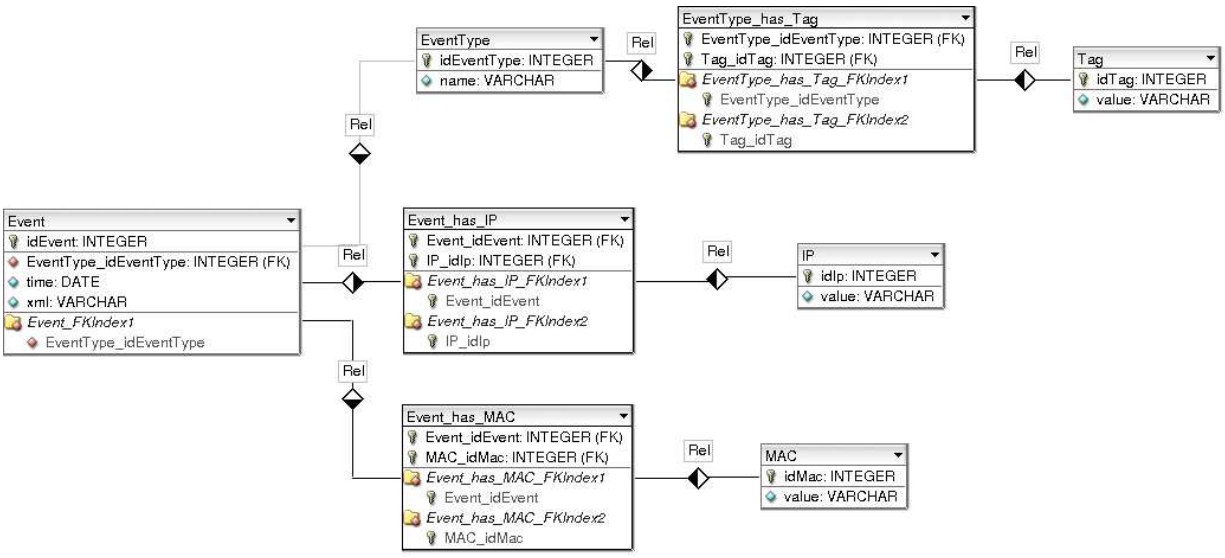

Fig. 3.6: Modelo do banco objeto-relacional para o serviço de *logging*.

No modelo objeto-relacional proposto, são utilizados parâmetros de busca nas tabelas para uma consulta inicial que retorna um conjunto de documentos XML, por exemplo, aqueles que possuem um determinado endereço MAC em um intervalo de tempo determinado. Consultas mais refinadas podem ser feitas aplicando-se filtros XPath a estes documentos. Desta forma é possível fazer uma consulta prévia de eventos para diminuir o espaço de busca, e posteriormente aplicar o filtro XPath

[1] http://www.modis.ispras.ru/sedna/ .

[2] http://www.postgresql.org .

Banco de dados	Forma de Armazenamento	Linguagem	Nº Elementos	Tempo de Busca
PostgreSQL	Tabelas	Java/JDBC	10.000	50 ms
PostgreSQL	Tabelas	Java/Hibernate	10.000	300 ms
Sedna	Documento XML	Java/XML:DB[3]	10.000	300 ms

Tab. 3.1: Comparação entre os modelos de banco de dados.

em cada evento. Este procedimento tem um desempenho melhor conforme a qualidade da pré busca efetuada, que diminuirá o conjunto de eventos a serem aplicados o filtro XPath, que trata-se de um processo custoso.

Testes foram realizados utilizando-se a linguagem Java [33] em uma máquina com processador Intel Core Duo T2300 de 1.66 Ghz com 1 GB de memória. O tempo de consulta foi medido para os bancos PostgreSQL e Sedna, cada um deles com 10.000 documentos distintos. Em ambos os casos o tempo foi de aproximadamente 300 ms para retornar apenas um evento XML. No PostgreSQL, 250 ms se devem ao Hibernate [34], um sistema baseado em Java para o mapeamento de dados persistentes em objetos; já que utilizando-se apenas a API JDBC [35] a busca foi de aproximadamente 50 ms. Na Tabela 3.1 há uma comparação entre os dois modelos analisados.

Eventos de *logging* tendem a crescer ao longo do tempo. Desta forma, um banco de dados que suporte um crescimento contínuo é necessário. O serviço de *logging* pode operar com banco de dados relacional ou XML nativo, porém acreditamos que o banco de dados relacional seja mais robusto e escalável do que o XML nativo, devido a maturidade do código e ser um banco de dados bem estabelecido. Sendo assim, este foi o padrão definido para o serviço.

3.4 Serviço de Relatório

Como diferentes aplicações podem requerer consultas semelhantes ao serviço de *logging* foi proposto o serviço de relatório, que tem como principal objetivo prover relatórios pré-definidos, simplificando as consultas e deixando o cliente livre de conhecer a interface do serviço de *logging*.

Este serviço, portanto, tem por objetivo facilitar a recuperação de dados de *logging* e promover consultas ao *logging* de forma simples. Um relatório é definido como uma consulta pré-determinada, em que os parâmetros já estão configurados e, caso necessário, também é possível serem enviados parâmetros adicionais na requisição de um relatório.

O serviço de relatório recebe uma requisição do cliente, monta e envia uma consulta ao serviço de *logging*, recupera eventos armazenados e os estrutura na forma de um relatório que é devolvido ao cliente. A requisição é feita através de uma mensagem HTTP GET contendo os parâmetros de entrada, o tipo do relatório e, opcionalmente, informações adicionais de busca. A resposta é uma lista de eventos. Observe que o cliente não precisa fornecer a expressão XPath ou parâmetros para uma consulta, como é necessário acessando diretamente a interface do serviço de *logging*. O funcionamento de uma requisição de relatório pode ser verificado na Figura 3.7.

O serviço também disponibiliza uma interface para o cadastro de novos tipos de relatório e listagem e detalhamento dos tipos existentes. Para cada tipo de relatório criado devem ser definidos os parâmetros usados na requisição. Esta interface permite, ainda, a visualização de relatórios em HTML (*HyperText Markup Language*). Desta forma o cliente tem a opção de visualizar relatórios ou então obtê-los no formato XML. O armazenamento dos relatórios cadastrados é feito de forma

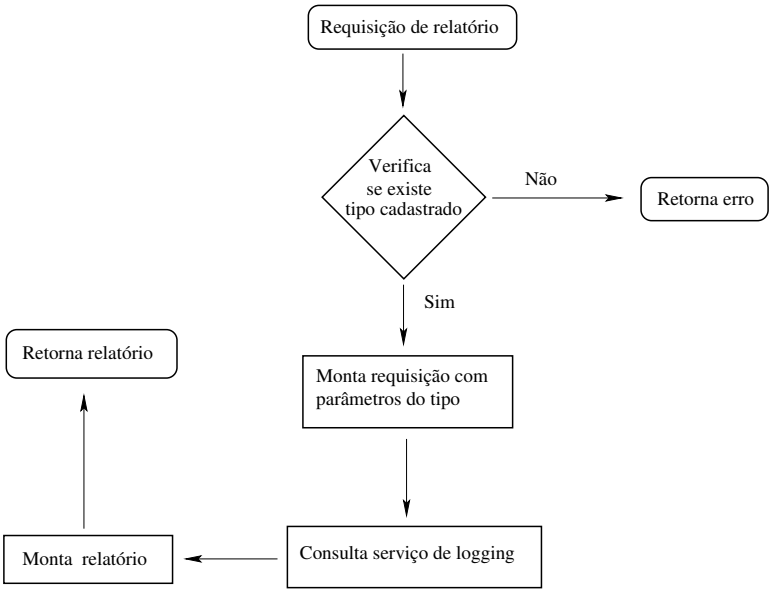

Fig. 3.7: Fluxograma requisição de relatório.

persistente em banco de dados.

O modelo de banco de dados para o serviço de relatório pode ser observado na Figura 3.8. No modelo proposto há uma tabela principal que se refere ao tipo do relatório. Esta tabela contém atributos referentes a este tipo, como o nome e descrição, os demais atributos serão utilizados como parâmetros para consulta ao serviço de *logging*, como tipo de evento e filtro XPath. A tabela de tipo de relatório se relaciona com outras tabelas que contém valores IP, MAC e outras marcas. Estes valores também poderão ser utilizados em consultas ao serviço de *logging*.

3.5 Serviço de *Proxy*

O gerenciamento de recursos existentes na rede, como roteadores, switches e serviços, é parte fundamental no âmbito da gerência de redes, mobilidade e engenharia de tráfego, pois é necessário um monitoramento e atuação nestes recursos, conforme o estado em que se encontre a rede. Para este fim tem-se o serviço de *proxy*, que centraliza o acesso a estes recursos através de agentes que serão responsáveis por atuar e monitorar tais recursos. Estes agentes estão descritos com um maior nível de detalhes na Seção 3.5.1.

Através do serviço de *proxy*, o acesso aos agentes pelas aplicações clientes é feito de forma transparente, sendo que o serviço funciona como um intermediário repassando a requisição ao agente alvo e retornando a resposta ao cliente. Desta maneira, um cliente não necessita ter acesso diretamente ao agente de recurso, que pode estar localizado em uma rede com acesso restrito. Ao centralizar o acesso, a segurança dos recursos também é facilitada, e será provida por meio do serviço AAA (*Authentication Authorization Accounting*), descrito na Seção 3.6

O serviço de *proxy* também fornece uma lista dos agentes e serviços disponíveis com uma descrição de cada um deles. A Figura 3.9 apresenta a interação das aplicações e agentes com o serviço de *proxy* e informações sobre agentes mantidas pelo serviço. Uma aplicação acessa o serviço de *proxy*,

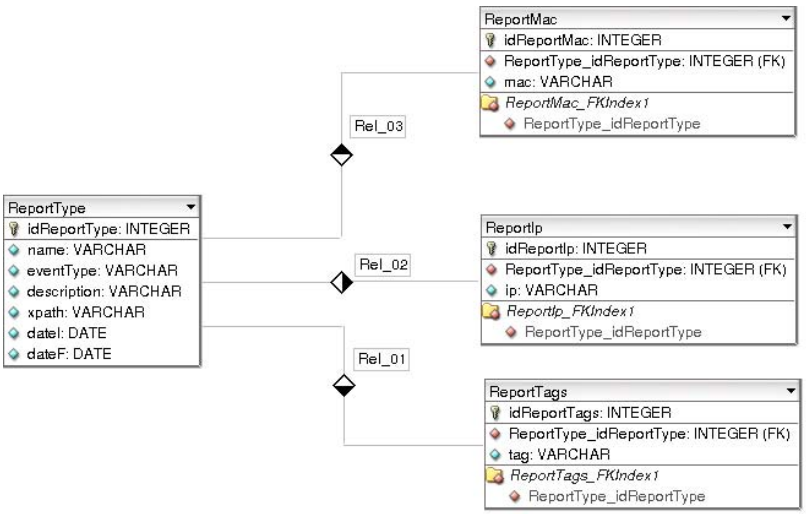

Fig. 3.8: Modelo do banco de dados para o serviço de relatório.

e então o serviço passa a requisição para o agente específico retornando a resposta para a aplicação. Uma aplicação também pode verificar quais agentes estão ativos, enviando uma requisição ao serviço de *proxy*, que retorna uma lista com detalhes sobre cada agente. Os agentes ativos no serviço de *proxy* necessitam ser registrados. Este registro é feito por meio de uma mensagem HTTP POST, contendo informações do agente, como a sua URI, detalhes dos serviços que ele é capaz de prover e uma descrição referente ao agente.

O serviço de *proxy* mantém informações e serviços referentes aos agentes que foram registrados de forma *soft-state*, o que exige um registro periódico de agentes. O serviço mantém um contador temporal e periodicamente verifica e remove os agentes que tiverem seu tempo de permanência expirado. Ao ser efetuado o registro de um agente, o contador referente ao agente será reiniciado. Desta forma, as requisições de registro no serviço de *proxy* funcionam como mensagens de *keepalive*[4] mantendo o estado ativo dos agentes e serviços.

Os serviços disponibilizados pelo serviço de *proxy* e por agentes de recurso são expressos na forma de uma URI específica, como ilustrado na Figura 3.9. Este padrão de comunicação está detalhado na Seção 3.5.4. O armazenamento dos agentes ativos no serviço de *proxy* é feito através de uma estrutura de árvore visando uma busca por serviços eficaz. Esta estrutura de armazenamento no serviço de *proxy* foi denominada árvore de recursos. A Figura 3.10 ilustra como são armazenados os serviços e agentes nesta árvore. De maneira geral, ao ser buscado um serviço expresso por uma URI, deve ser retornado o agente que disponibiliza este serviço. Caso não exista nenhum agente cadastrado para um serviço, deverá ser retornado o agente que tenha um serviço disponibilizado que mais se aproxime hierarquicamente da URI que está sendo buscada. Por exemplo, o "Agente A" disponibiliza um serviço `/routers/dca01/interfaces` e têm-se um cliente busque pelo serviço `/mis/proxy/routers/dca01/interfaces` no serviço de *proxy*, a requisição deverá ser encaminhada para o "Agente A". Caso um cliente busque pelo serviço

[4]Sinal enviado em intervalos pré-definidos, caso não haja resposta o agente é tido como inativo.

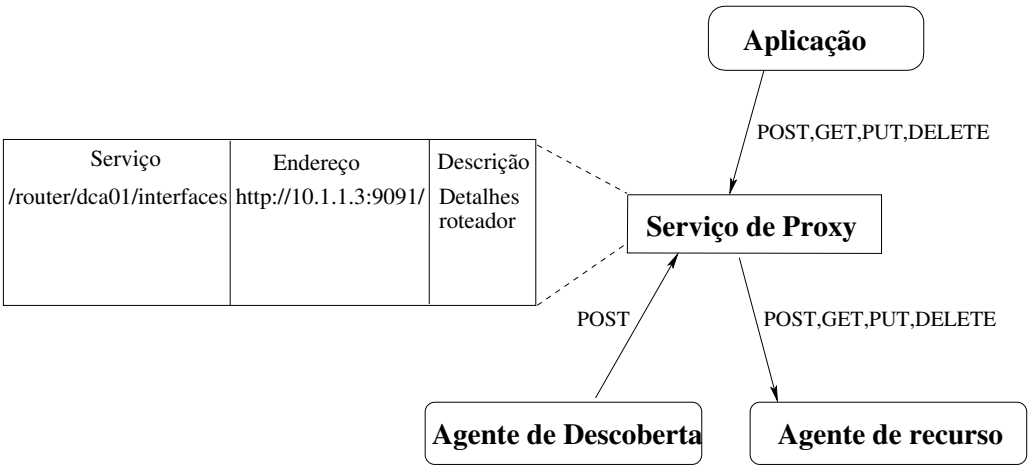

Fig. 3.9: Serviço de *proxy*.

`/mis/proxy/routers/dca01/interfaces/eth0/tc/class`, e este serviço não exista registrado no serviço de *proxy*, a requisição deverá ser encaminhada para o "Agente A", que na hierarquia da URI é o que mais se aproxima. Hierarquias de recurso são similares ao esquema de identificação de objetos de gerência em MIBs (*Management Information Base*) SNMP [30], exceto que números são substituídos por nomes simbólicos.

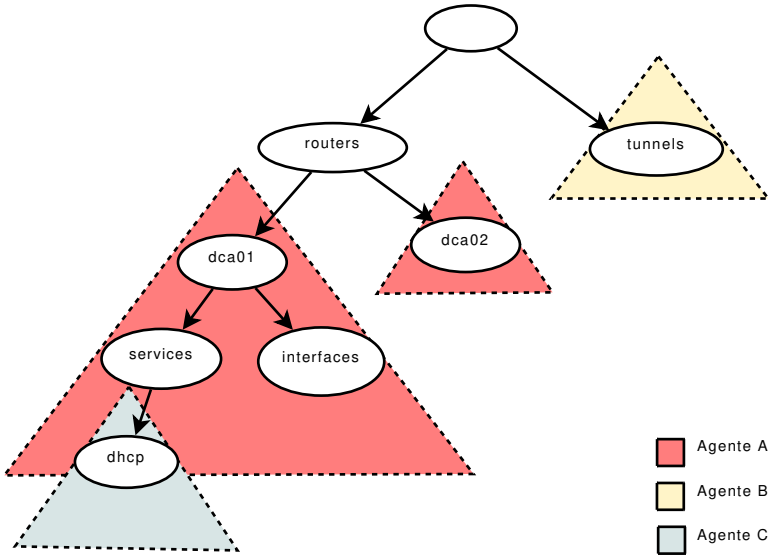

Fig. 3.10: Árvore de recursos do serviço de *proxy*.

3.5.1 Agentes de Recurso

Os agentes de recurso apresentam a capacidade de coletar informações de um ou mais recursos ou serviços da rede e alterar seus estados e configurações. Estes agentes são compostos por duas partes

principais: sensor e atuador, responsáveis por ler e alterar o estado e configurações de um recurso específico.

Um agente além de controlar um recurso, pode requisitar outro agente em uma determinada ação que necessite realizar. Sendo assim, um agente pode ser acessado pelo serviço de *proxy* ou por outros agentes. A Figura 3.11 mostra a relação entre os agentes e o serviço de *proxy*.

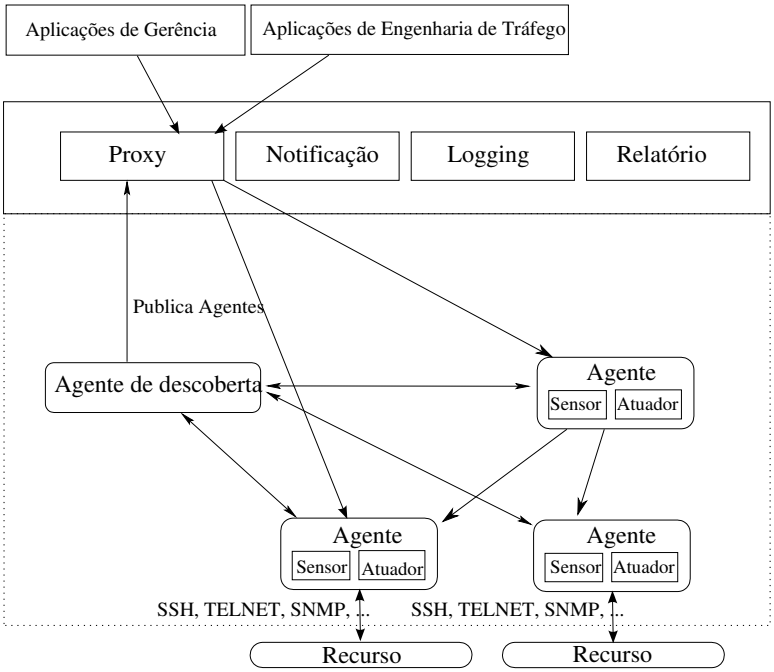

Fig. 3.11: Agentes e serviço de *proxy*.

Os agentes de recurso necessitam ter acesso aos recursos que irão controlar, podendo estar localizados no próprio recurso, ou em um nó que tenha acesso à rede de gerência da arquitetura. A comunicação dos agentes com os recursos pode ser feita com algum protocolo de controle específico, como por exemplo SSH [36], SNMP [30], Telnet [37], dependendo da implementação do agente. Já a comunicação dos clientes (via serviço de *proxy*) com os agentes de recurso é feita utilizando o modelo REST (*Representational State Transfer*) [38], descrito na Seção 3.5.3. Este modelo apresenta características que se mostraram relevantes para forma de acesso aos recursos. A utilização do modelo REST no serviço de *proxy* e agentes está descrita na Seção 3.5.4.

3.5.2 Agente de Descoberta

Os agentes de recurso em atividade necessitam ser registrados junto ao serviço de *proxy* para que este possa armazenar em sua árvore de recursos quais serviços e agentes podem ser acessados pelos clientes. Um agente especial denominado agente de descoberta é capaz de descobrir os agentes ativos na rede e fazer o registro deles junto ao serviço de *proxy*. A descoberta pode ser feita por *polling* nos agentes, ou seja, requisições são enviadas periodicamente aos agentes de forma que quando um agente responde, este é marcado como ativo. Um outro método para descoberta pode ser baseado

nos agentes, em que o agente é capaz de enviar mensagens de *keepalive* para o agente de descoberta indicando que o mesmo encontra-se ativo.

Um agente de recurso ao ser iniciado deverá enviar uma requisição de registro junto ao agente de descoberta que irá armazenar os agentes existentes no sistema. O registro de um agente de recurso é constituído pelos seguintes campos:

- URI referente ao agente;
- Serviços disponibilizados;
- Descrição do agente.

O registro inicial de um agente é necessário, pois o agente de descoberta irá armazenar todos os agentes existentes na rede para que em um segundo momento possa verificar a atividade dos mesmos e, assim, registrá-los no serviço de *proxy*. Este registro deve ser persistente para que caso o agente de descoberta cesse a execução por algum motivo, ao ser reiniciado possa recuperar informações dos agentes de recurso que tinham se registrado anteriormente. Desta forma, este agente apresenta tolerância e recuperação a possíveis falhas.

3.5.3 Arquitetura REST

REST (*Representational State Transfer*) é um estilo de arquitetura de software para sistemas de hypermidia distribuída. Foi proposto por Roy Fielding [38], e é baseado na arquitetura Web. Os princípios básicos do REST são:

- Cliente-servidor: necessidade uma arquitetura que seja baseada no modelo cliente-servidor;
- Sem estado: a interação cliente-servidor deve ser sem estado, ou seja, cada requisição feita ao servidor deve conter toda informação necessária para entendê-la, não sendo necessário gravar nenhum estado das comunicações;
- *Cache*: é utilizado para melhorar a eficiência eliminando algumas possíveis interações. Uma resposta armazenada em *cache* pode ser reutilizada posteriormente caso uma requisição equivalente seja feita. O *cache* requer que os dados contidos na resposta para uma requisição sejam marcados para serem mantidos em *cache* ou não;
- Interface uniforme: REST enfatiza a existência de uma interface comum entre componentes. Esta interface segue os conceitos de generalidade da engenharia de software sendo que todas as interações de componentes seguem um mesmo padrão, tendo um conjunto de operações bem definidos que vão ser aplicados a todos os recursos de informação;
- Sistema em camadas: permite que uma arquitetura seja composta de diversas camadas hierárquicas respeitando o comportamento dos componentes, de tal forma que um componente não possa ter conhecimento da camada seguinte à que eles estão interagindo. Um sistema de camadas melhora a escalabilidade e é utilizado para encapsular serviços legados, ou proteger novos serviços. Permite a introdução de pontos intermediários na comunicação, sem que seja necessário mudar as interfaces dos componentes, melhorando a performance em grande escala.

REST propõe uma abstração dos elementos arquiteturais no sistema, ignorando detalhes de implementação de componentes e sintaxe de protocolo tendo como objetivo focar na função dos componentes e nas interações com outros componentes. O comportamento dos elementos no REST é fundamentado em aplicações baseadas na rede.

Recurso é o elemento principal de abstração de informação no REST. Qualquer tipo de informação que pode ser nomeada pode ser considerada um recurso, como, por exemplo, um serviço disponibilizado, um arquivo de texto, uma imagem, ou um conjunto de outros recursos. Cada recurso possui um identificador específico que é utilizado nas interações entre componentes. A abstração de elementos de informação em recurso permite que eles sejam considerados fontes de informação, sem diferenciação por tipo de implementação. É possível ainda ter-se diversas representações para um recurso, eventualmente com negociação de conteúdo baseado nas características da requisição feita para este recurso.

Representações em REST capturam o estado de um recurso, ou seja, informações que representam determinado recurso. Um formato de dados de uma representação é conhecido como tipo de mídia.

Em REST conectores são interfaces abstratas para a comunicação com um componente. Todas as interações são sem estado, ou seja, cada requisição contém toda informação que é necessária para um conector entendê-la, não importando quantas requisições foram processadas anteriormente. Uma aplicação pode acessar um recurso conhecendo apenas o seu identificador e a ação desejada. Uma requisição pode passar por diversos conectores (servidores, caches, *proxy*, clientes) que existam entre o cliente e o servidor que mantém a informação. A aplicação deve conhecer o formato da informação da resposta, a representação, que pode ter sido especificado na requisição.

3.5.4 REST na comunicação de agentes e serviço de *proxy*

De maneira geral o modelo REST pressupõe que cada recurso possui um identificador expressado por sua URI; um conjunto de operações bem definidas que se aplicam genericamente a todos os recursos; um protocolo cliente-servidor sem estado, ou seja, cada requisição contém toda informação necessária para ser compreendida e o uso de hipermídia, como conteúdo de informação e transições de estado. No caso da plataforma, o protocolo a ser utilizado é o HTTP, as operações são definidas pelos métodos padrão do HTTP (POST, GET, PUT, DELETE). As informações trafegam no formato XML da mesma forma que nos outros serviços. Cada agente e serviço disponível por ele é identificado por uma URI específica. A Tabela 3.2 faz uma associação entre os métodos HTTP e possíveis operações a serem executadas. Os elementos nos recursos podem ser configurações, como por exemplo o endereço IP de uma interface de rede em um roteador, ou também instâncias, no caso do serviço de *proxy* os agentes de recurso ativos, armazenados na árvore de recursos.

Método HTTP	Operação associada
GET	Ler estado de um recurso
POST	Criar um elemento no recurso
PUT	Atualizar um elemento no recurso
DELETE	Apagar um elemento no recurso
OPTIONS	Listar informações de representação

Tab. 3.2: Métodos HTTP e operações associadas.

A utilização do modelo REST e o esquema de identificação por URI permite uma nomeação de recursos mais abstrata, diferente da identificação de objetos em MIBs SNMP que é numérica. Desta maneira, ao se identificar cada recurso por uma URI é simplificado o acesso aos recursos e a granularidade da informação.

Por exemplo: se um cliente deseja verificar qual o endereço de uma interface do roteador *dca10*, a URI `http://mis/proxy/routers/dca10/interfaces/eth0` poderia retornar informações pertinentes a interface *eth0* deste roteador, como endereço MAC, IP, etc. Neste mesmo exemplo, se fosse acessado a URI `http://mis/proxy/routers/dca10/interfaces`, poderiam ser obtidos dados de todas as interfaces presentes no roteador *dca10*. Pode-se notar que na própria hierarquia da URI é possível se obter o acesso a um recurso mais específico ou mais genérico.

Os métodos HTTP são utilizados conforme a ação desejada em um determinado recurso ou serviço. De forma semântica, requisições do tipo GET indicam a obtenção de informação, ou leitura de dados; do tipo POST indica a inserção de dados; do tipo PUT a alteração de dados já inseridos; do tipo DELETE a remoção de dados. Desta forma, todos os recursos possuirão um padrão para as ações e utilização delas usufruindo do padrão do próprio protocolo sem que se seja necessária qualquer alteração.

3.6 Serviço AAA

O serviço autorização, autenticação e contabilidade ou AAA é responsável pelo acesso seguro à plataforma MIS, permitindo que apenas clientes autorizados tenham acesso aos serviços e recursos. Toda requisição antes de atingir determinado serviço passa pelo serviço AAA que irá bloquear ou permitir o acesso dependendo das informações de identidade do cliente. Desta forma, este serviço é intermediário e obrigatório para toda requisição à plataforma MIS.

O serviço AAA constitui uma camada de segurança na plataforma que além de permitir ou negar acesso também pode ser responsável pela criptografia de dados trafegados, visando uma segurança maior nas transações entre os clientes e a plataforma.

Um cliente ao acessar algum serviço ou recurso da plataforma MIS deverá se autenticar previamente. Para isto, deve fornecer uma credencial, que pode ser um certificado digital, uma chave ou usuário e senha. O serviço AAA, ao receber esta credencial, efetua a autenticação do cliente, que em caso negativo retorna um erro de autenticação e em caso positivo irá verificar a permissão do cliente para o recurso que ele deseja acessar. Caso o cliente não tenha permissão para acessar o recurso é enviado um erro de autorização, caso contrário é liberado o acesso a este recurso. O funcionamento do serviço AAA é ilustrado pelo fluxograma da Figura 3.12.

Os acessos à plataforma MIS podem ser controlados pelo serviço AAA e um *logging* pode ser mantido de quais clientes acessaram a plataforma caso seja necessário algum tipo de auditoria. O serviço AAA também será utilizado por uma interface de gerência da plataforma MIS, descrita na Seção 3.7, já que um cliente ao acessar esta interface também deverá ser autenticado e autorizado.

O cliente do serviço AAA pode ser um administrador da rede fazendo o acesso via *Web browser* ou uma aplicação. Além disso, há a possibilidade de uma lista de controle de acesso (ACL - *Access Control List*) ser consultada para determinar quais operações são permitidas ao cliente.

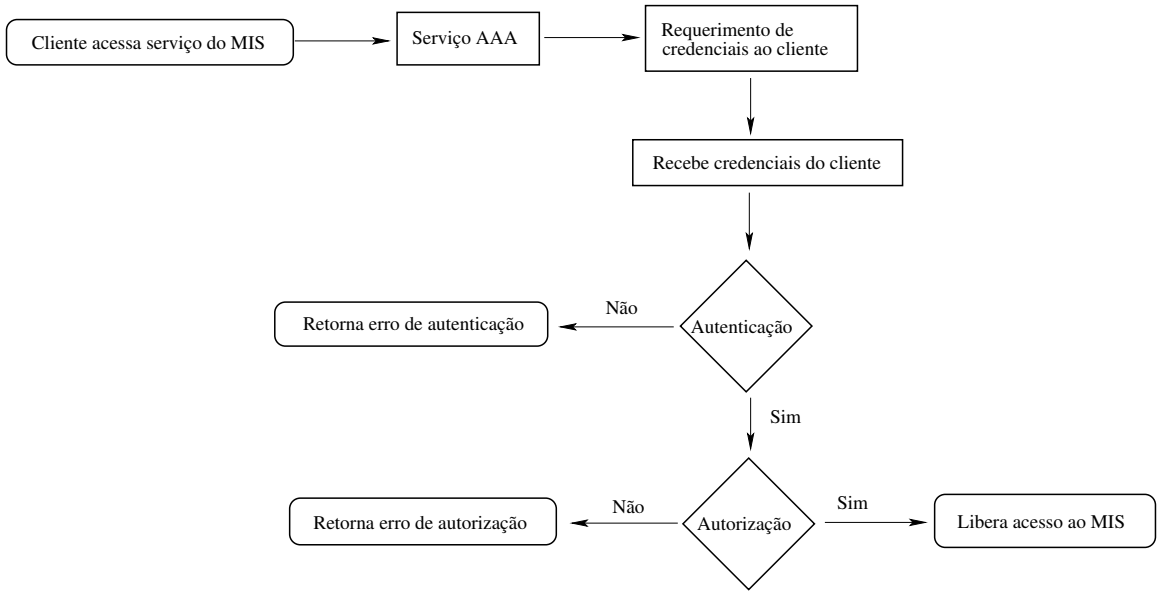

Fig. 3.12: Fluxograma do Serviço AAA.

3.7 Interface de Gerência

A gerência dos serviços presentes na plataforma MIS é importante para controlar e verificar seus estados, ter acesso as funcionalidades providas por eles bem como facilitar a realização de testes. Neste sentido é necessário uma interface Web para a gerência dos serviços MIS que permita ser acessada facilmente por um navegador Web em qualquer ponto da rede que possua acesso a plataforma MIS. Por meio dessa interface é possível visualizar, enviar, modificar e remover dados de cada serviço disponibilizado pela plataforma. Um menu poderá ser utilizado para navegar entre os serviços e selecionar as opções em cada um deles.

No serviço de notificação no modelo *pull* a interface permite efetuar busca por eventos, listar todos os eventos ativos e remover um evento. Pode-se, também, efetuar o registro de clientes no serviço de notificação (modelo *push*), obter uma lista dos clientes registrados, modificar e remover os registros. É possível também, efetuar o envio de um evento para o serviço, sendo importante para realização de testes tanto no próprio serviço quanto nas aplicações que dele fazem uso.

A interface de gerência possibilita uma consulta ao serviço de *logging* provendo uma tela que permite o envio de uma requisição de consulta ao serviço de *logging*, com todos os parâmetros disponíveis. A resposta é exibida em formato HTML para o cliente.

No serviço de relatório todas as funcionalidades podem ser acessadas pela interface Web de gerência. O cliente poderá efetuar operação de cadastro de novos relatórios de acordo com parâmetros como tipo de evento ou marcas no evento. Estas informações serão buscadas no banco de dados do serviço de *logging* e exibidas para o cliente no momento do cadastro para que o cliente selecione opções que já estejam armazenadas no serviço de *logging*, como por exemplo tipos de eventos já armazenados, facilitando a tarefa de cadastramento de relatório.

Ainda no serviço de relatório, uma listagem dos tipos de relatório já cadastrados pode ser obtida, permitindo uma visualização da resposta gerada por cada tipo de relatório no formato HTML, bem

como a modificação dos parâmetros e remoção de um tipo cadastrado. Neste serviço também é possível gerar um relatório em tempo de execução, sem que seja necessário criar um tipo e salvá-lo, sendo necessário apenas fornecer os parâmetros de busca.

No serviço de *proxy* a interface de gerência tem a funcionalidade do controle dos agentes de recurso ativos no serviço, permitindo listagem dos agentes registrados, exibição de descrições destes agentes, tempo de permanência deles no serviço, modificação e remoção dos mesmos.

O capítulo a seguir apresenta a modelagem e implementação dos serviços da plataforma MIS descritos neste capítulo.

Capítulo 4

Modelagem e implementação da plataforma

O desenvolvimento da plataforma MIS é descrito detalhadamente neste capítulo. Primeiramente é apresentada uma modelagem estrutural e comportamental da plataforma e dos serviços disponibilizados. Posteriormente são apresentados informações, detalhes e tecnologias utilizadas na implementação do modelo de serviço proposto. Por fim, são descritos os testes realizados com a plataforma implementada.

4.1 Modelagem UML

A modelagem da plataforma MIS emprega a linguagem de modelagem de software UML [39] (*Unified Modelling Language*).

UML é uma linguagem visual, isto é, toda modelagem do software pode ser representada de forma gráfica. UML tem com objetivo fornecer uma maneira comum de capturar e expressar relações, comportamentos e idéias de um nível mais alto na forma de notações que sejam simples e eficientes de escrever e entender [40].

UML em sua versão 2.0 define duas categorias de diagramas: estrutural e comportamental, que, respectivamente, definem a arquitetura e organização estática do sistema e as interações e comportamento dos elementos do sistema [40]. Desta maneira, a modelagem da plataforma MIS seguirá este padrão fazendo uso de diagramas para abordar modelos estruturais e comportamentais.

4.1.1 Diagramas de Casos de Uso

Os diagramas de casos de uso da linguagem UML capturam as funcionalidades e requisitos do sistema. Estes diagramas são utilizados para descrever cenários de utilização do sistema, neste caso, a plataforma MIS e seus serviços. Os casos de uso apresentam uma visão de alto nível do comportamento dos serviços, funcionalidades e utilização da plataforma MIS e através deles podem ser analisados os requisitos de cada serviço disponibilizado. Os principais casos de uso da plataforma MIS são detalhados a seguir.

Serviço de notificação: modo *push*

As funcionalidades do serviço de notificação no modo *push* e o envio de eventos podem ser observados no diagrama de casos de uso da Figura 4.1. Os atores, neste caso, são um produtor de eventos e um consumidor de eventos. Um consumidor de eventos pode se registrar para receber eventos de seu interesse. O serviço de notificação armazena informações referentes a este consumidor de eventos. Um produtor de eventos pode enviar um evento ao serviço de notificação. O serviço, ao receber o evento, pode enviá-lo a um consumidor de eventos registrado caso o evento seja de interesse desse consumidor de eventos. Este mesmo evento é armazenado no serviço de notificação para ser consultado no modo *pull*.

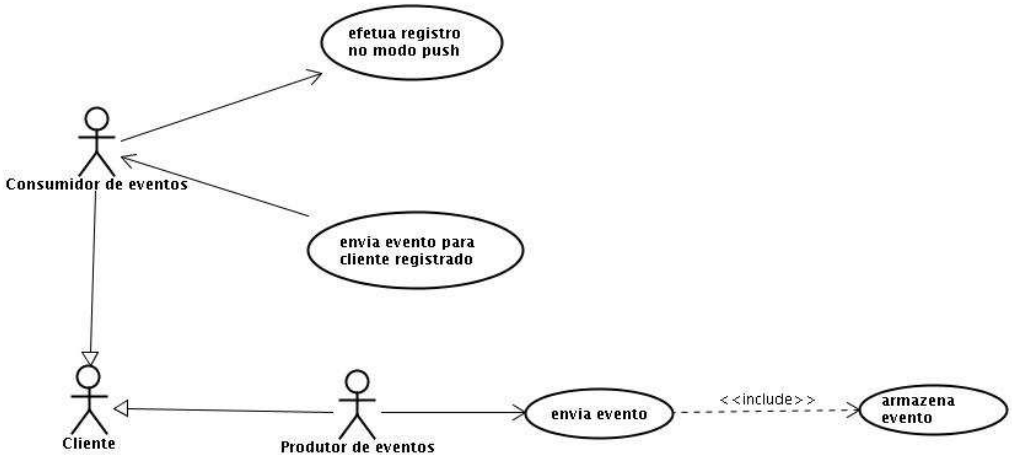

Fig. 4.1: Casos de uso modo *push* do serviço de notificação.

Serviço de notificação: modo *pull*

As principais funcionalidades no modo *pull* do serviço de notificação podem ser observadas no diagrama de casos de uso da Figura 4.2. Neste caso, um consumidor de eventos efetua uma requisição de busca no serviço, que por sua vez recupera eventos armazenados que sejam compatíveis com a requisição feita. De posse destes eventos, o serviço de notificação monta uma resposta retornando-a ao cliente.

Fig. 4.2: Casos de uso modo *pull* do serviço de notificação.

Serviço de notificação: interação com o serviço de *logging*

Os casos de uso no envio de evento para *logging* podem ser observados no diagrama da Figura 4.3. Os atores são um produtor de eventos e o serviço de *logging*. Um produtor de eventos, submete um evento marcado para armazenamento em *logging* ao serviço de notificação. Este serviço verifica o evento e o encaminha para o serviço de *logging*. O serviço de *logging* efetua ações necessárias para armazenar o evento no banco de dados.

Fig. 4.3: Casos de uso do envio de evento para *logging*.

Serviço de *logging*

Os casos de uso para consulta no serviço de *logging* podem ser observados no diagrama da Figura 4.4. O ator é um cliente do serviço. O cliente efetua uma requisição de consulta ao serviço de *logging* passando os parâmetros necessários. Este serviço faz uma consulta no banco de dados obtendo eventos armazenados e monta uma resposta no formato especificado pela requisição.

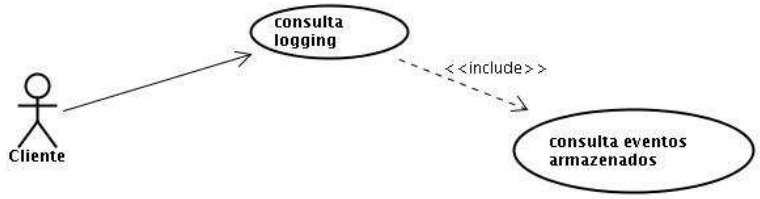

Fig. 4.4: Casos de uso de consulta no serviço de *logging*.

Serviço de relatório

Os casos de uso do serviço de relatório podem ser observados no diagrama da Figura 4.5. Os atores são um cliente e o serviço de *logging*. Um cliente poderá requisitar um relatório ao serviço de relatório que irá consultar o tipo de relatório no banco de dados recuperando parâmetros de busca. De posse destes dados o serviço de relatório enviará uma requisição de consulta ao serviço de *logging* e retornará uma resposta ao cliente com os dados obtidos no serviço de *logging*. Um cliente também poderá cadastrar, consultar ou alterar tipos de relatório. O serviço de relatório irá consultar ou armazenar estes tipos no banco de dados conforme a ação desejada.

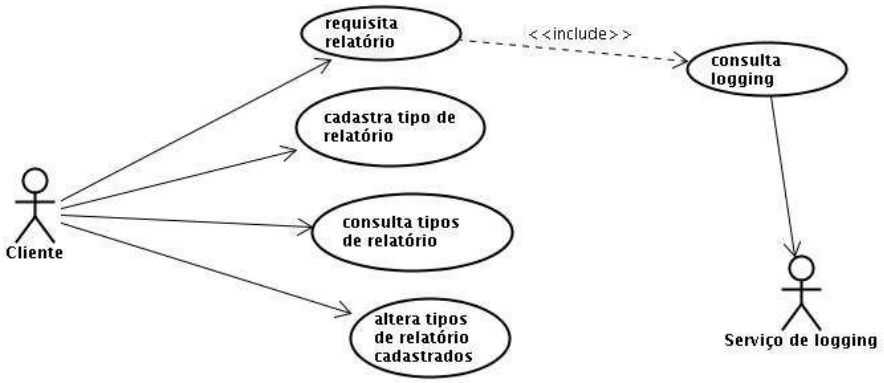

Fig. 4.5: Casos de uso do serviço de relatório.

Serviço de *proxy*

Os casos de uso de acesso ao serviço de *proxy* podem ser observados no diagrama da Figura 4.6. Os atores são um cliente e um agente de recurso. Um cliente poderá acessar um recurso enviando uma requisição ao serviço de *proxy* que irá verificar qual agente mantém o recurso que o cliente deseja acessar, redirecionando a requisição para o agente que provê tal recurso. Esse agente, ao receber a requisição, irá acessar o recurso em questão. Um cliente também poderá requisitar uma lista de quais são os agentes ativos no serviço de *proxy*.

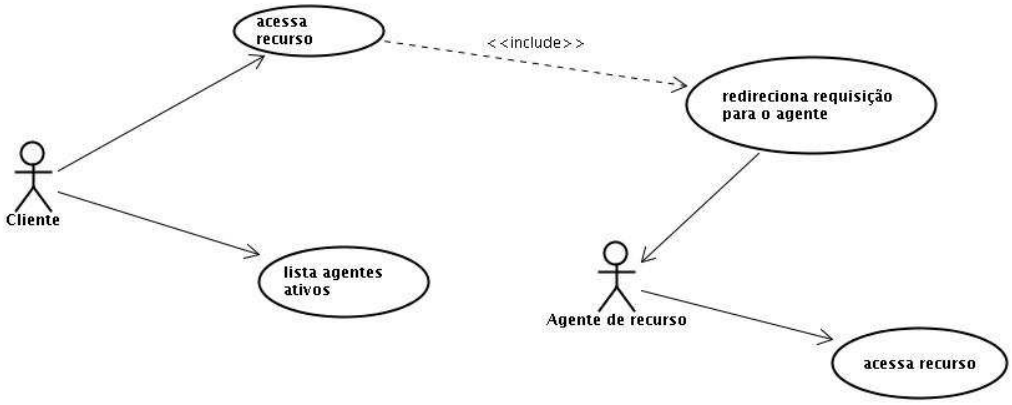

Fig. 4.6: Casos de uso de acesso ao serviço de *proxy*.

Registro de agentes de recurso e Agente de Descoberta

Os casos de uso para o registro de agentes de recurso junto ao serviço de *proxy* e no agente de descoberta, podem ser observados no diagrama da Figura 4.7. Os atores são o agente de descoberta e um agente de recurso. O agente de descoberta é responsável por efetuar o registro de agentes de recurso ativos junto ao serviço de *proxy*. Para isto, um agente de recurso deve efetuar um registro ao ser iniciado junto ao agente de descoberta, este por sua vez irá armazenar o agente e também poderá recuperar agentes que foram previamente registrados. O agente de descoberta irá verificar

periodicamente quais agentes estão ativos no sistema. Os agentes que estiverem ativos permanecerão registrados junto ao serviço de proxy.

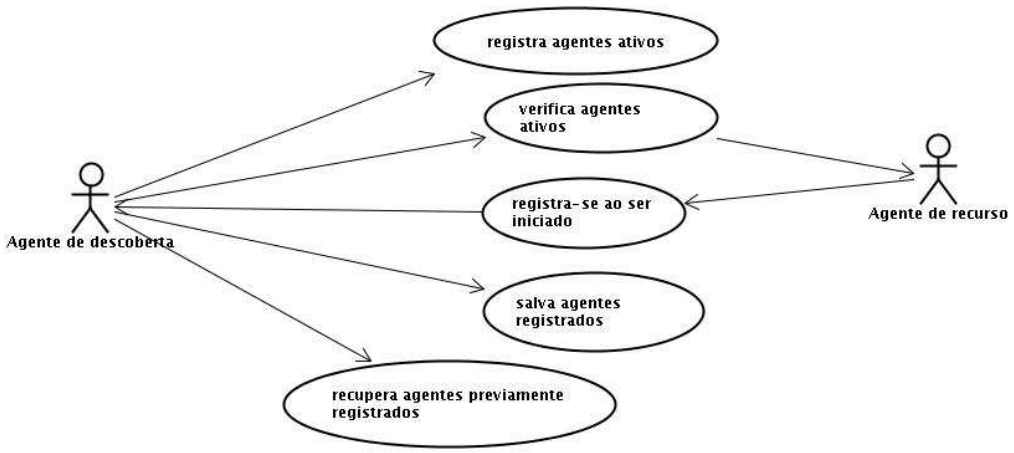

Fig. 4.7: Casos de uso de registro de agentes de recurso e agente de descoberta.

Casos de uso do serviço AAA

Os casos de uso do serviço AAA podem ser observados no diagrama da Figura 4.8. O ator é um cliente do serviço. Um cliente ao acessar a plataforma MIS deve se autenticar previamente no serviço AAA enviando suas credenciais junto à requisição de acesso. O serviço AAA, por sua vez, deve inspecionar as credenciais fornecidas pelo cliente, permitindo o acesso à plataforma caso estejam corretas. Caso contrário, o serviço retorna um erro de autenticação e bloqueia a requisição.

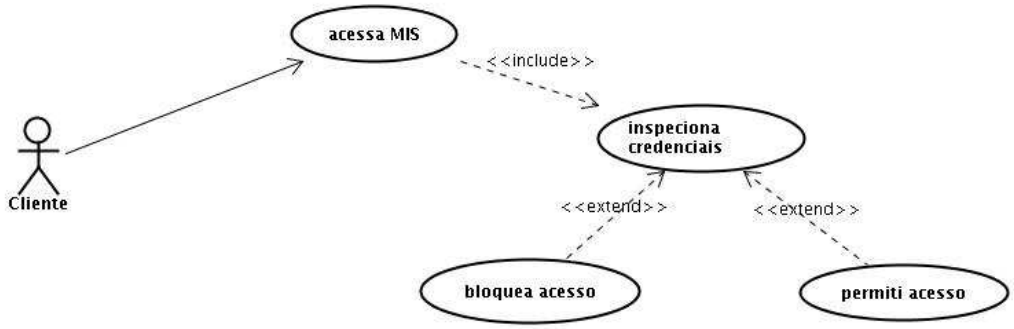

Fig. 4.8: Casos de uso do serviço AAA.

Casos de uso da interface de gerência da plataforma MIS

Os casos de uso da interface de gerência da plataforma MIS podem ser observados no diagrama da Figura 4.9. O ator neste caso é um gerente, que poderá gerenciar todos os serviços providos pela

plataforma MIS. Em cada serviço é disponibilizado um conjunto de tarefas específicas para gerenciá-lo. No serviço de notificação será possível efetuar a busca, listagem e remoção de eventos. Também é possível registrar um cliente para o modo *push* do serviço e listar os clientes registrados. No serviço de *logging* será possível enviar um evento para *logging* ou efetuar uma consulta. No serviço de relatório é possível visualizar um relatório, cadastrar, listar e alterar tipos de relatório. No serviço de *proxy* é possível visualizar, alterar e remover os agentes de recurso e serviços ativos. No serviço AAA o gerente poderá inserir, alterar ou remover um usuário da plataforma MIS.

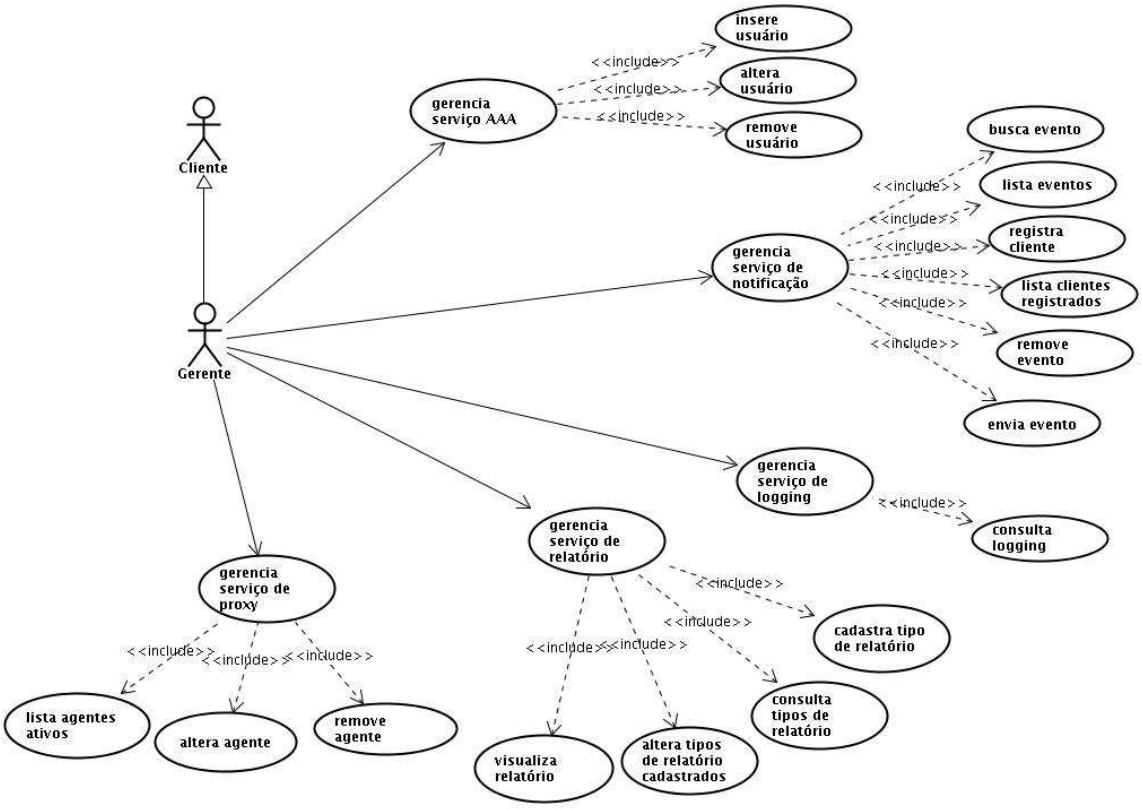

Fig. 4.9: Casos de uso da interface de gerência.

4.1.2 Diagramas de Classes

Os diagramas de classe representam uma visão estática de um sistema orientado a objetos, descrevendo também os atributos e métodos das classes. Estes diagramas são utilizados também para ilustrar as relações entre classes: herança, composição e conexões. No caso da plataforma MIS, os diagramas serão modelados por serviços, sendo apresentadas as classes e relações entre elas para cada serviço.

Serviço de notificação

O diagrama de classes para o serviço de notificação está ilustrado na Figura 4.10 e as principais classes definidas são:

- *NotificationServiceServlet*: esta classe é responsável por receber as requisições HTTP no serviço de notificação e invocar as devidas operações da classe *NotificationService*, como o registro de cliente no modelo *push*, envio de eventos e busca de eventos no modelo *pull*;

- *NotificationService*: é a classe principal do serviço de notificação e contém os métodos principais do serviço para o modo *push* e *pull*. Está presente nesta classe uma lista de clientes, instanciados pela classe *Subscriber* e uma lista e eventos instanciados pela classe *EventMessageBean*. Nesta classe é iniciado uma instância da classe *TimeChecker* para checagem de TTL das mensagens de eventos;

- *Notify*: esta classe é responsável por notificar clientes subscritos de acordo com o objeto instanciado da classe *Subscriber*, passado como parâmetro;

- *Subscriber*: esta classe representa um cliente subscrito no serviço de notificação. Contém como atributos a URI do cliente, que se refere ao endereço que ele recebe as notificações; um filtro XPath, que se refere ao tipo de evento que ele deseja receber e, também, um TTL, que representa o tempo que ele deseja receber notificação de eventos;

- *EventMessageBean*: esta classe representa um evento no serviço de notificação, ela contém um atributo que representa o documento XML do evento, um TTL que representa o tempo de permanência em memória no serviço de notificação e o tipo do evento;

- *TimeChecker*: esta classe é responsável por verificar o tempo de permanência (TTL) dos eventos no serviço de notificação, removendo os que possuam este tempo expirado. Esta classe é instanciada e iniciada na classe principal do serviço de notificação.

Serviço de *logging*

O diagrama de classes para o serviço de *logging* está ilustrado na Figura 4.11 e as principais classes definidas são:

- *LoggingServiceServlet*: esta classe é responsável por receber as requisições HTTP no serviço de *logging* e por invocar as devidas operações da classe *LoggingService*, como o envio de eventos para *logging* e a busca de eventos;

- *LoggingService*: esta é a classe principal do serviço de *logging*, contém os principais métodos para o envio de eventos para *logging* e a busca por eventos. Esta classe se relaciona com a classe *LoggingParser* que efetua o processamento de eventos XML e também com as classes que estendem a classe *LoggingDao*, que são responsáveis pela persistência dos dados;

- *LoggingParser*: esta classe é responsável pelo processamento de eventos, requisições e respostas no formato XML para o serviço de *logging*;

- *LoggingDao*: esta classe é uma classe abstrata responsável pelos métodos de acesso aos dados persistentes no serviço de *logging*, permitindo inserir, buscar, atualizar e remover dados. As classes responsáveis pela persistência dos dados estendem a classe *LoggingDao*;

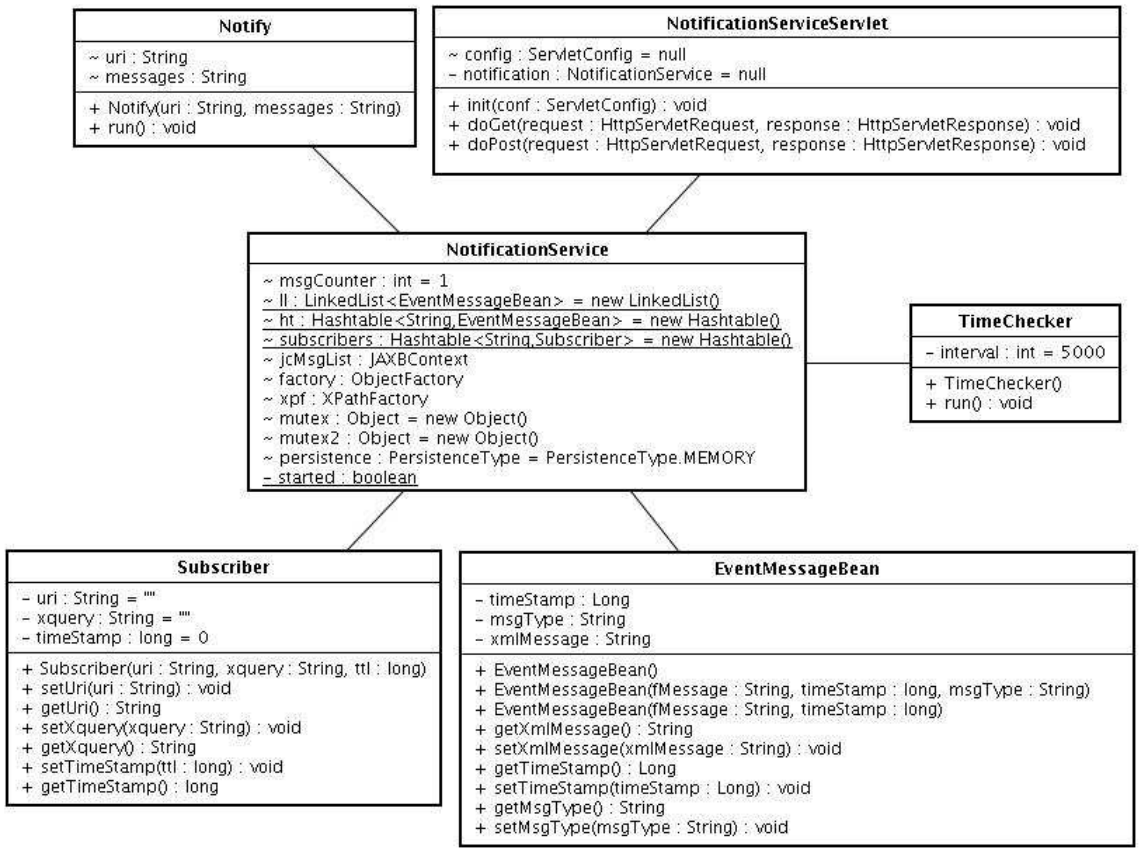

Fig. 4.10: Classes do serviço de notificação.

- *Event*: esta classe é uma entidade persistente que representa um evento, contém o documento XML e dados referentes ao evento, como tipo, data que foi salvo, endereços MAC e IP presentes e marcas existentes no evento;

- *LoggingQueryRequest*: esta classe representa uma requisição de busca feita ao serviço de *logging* contendo atributos utilizados em uma consulta ao serviço;

- *ParsedEvent*: esta classe representa um evento XML processado.

Serviço de relatório

O diagrama de classes para o serviço de relatório está ilustrado na Figura 4.12 e as classes definidas são:

- *ReportingServiceServlet*: esta classe é responsável por receber requisições HTTP do serviço de relatório e invocar as devidas operações da classe *ReportingService*, como a geração de relatório, cadastro de tipos, busca por tipos, alteração de tipos;

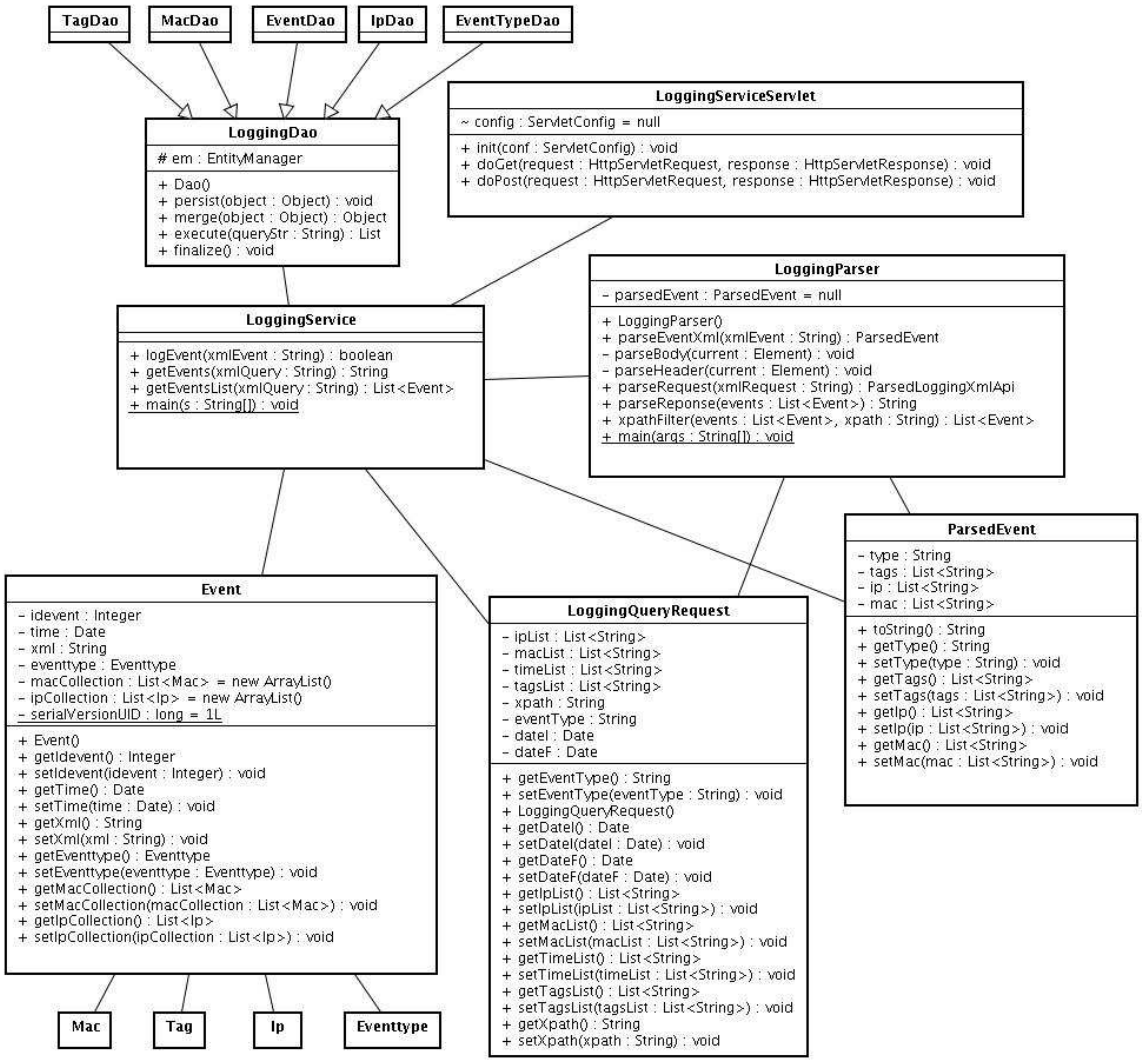

Fig. 4.11: Classes do serviço de *logging*.

- *ReportingService*: é a classe principal do serviço de relatório e apresenta os principais métodos para a geração de um tipo de relatório, cadastro, alteração e busca por tipos de relatório. Esta classe efetua comunicação com o serviço de *logging*;

- *ReportingParser*: é a classe responsável pelo processamento de eventos, requisições e respostas no formato XML para o serviço de relatório;

- *ReportTypeDao*: esta classe é responsável pelo acesso persistente aos tipos de relatório, efetuando inserção, busca, alteração e remoção de dados;

- *ReportType*: classe que representa um tipo de relatório com informações referentes ao tipo, como nome e também parâmetros de busca junto ao serviço de *logging*.

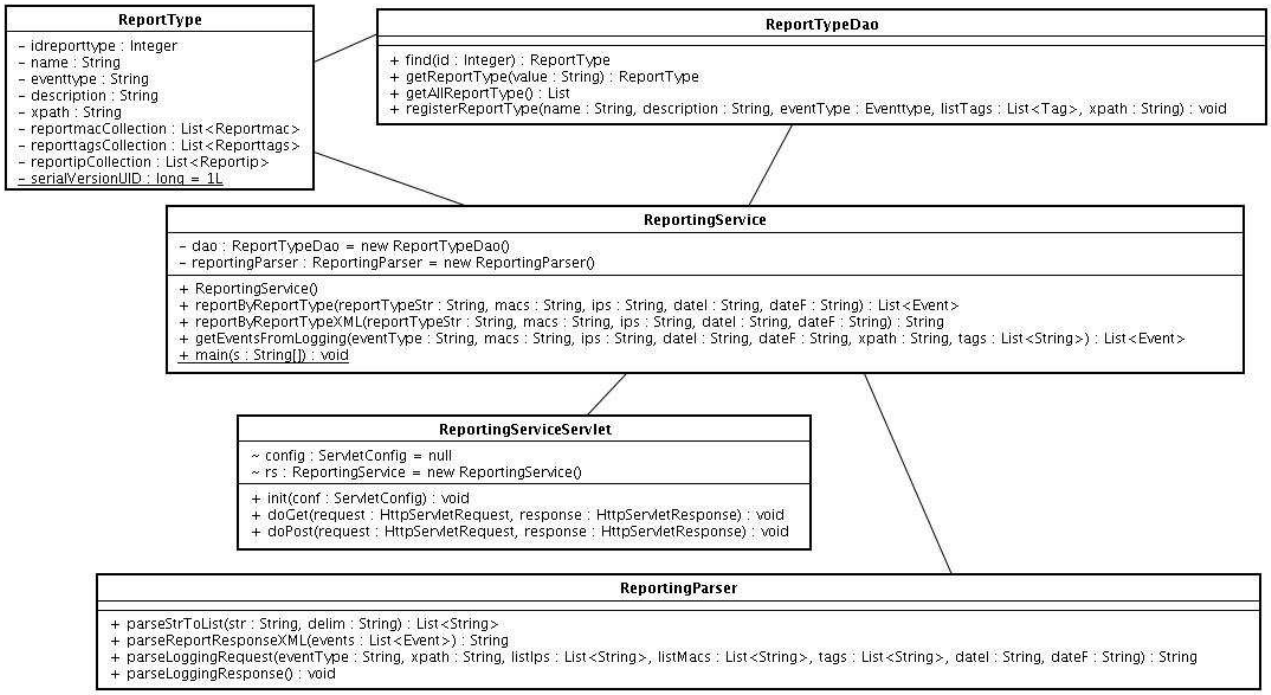

Fig. 4.12: Classes do serviço de relatório.

Serviço de proxy

O diagrama de classes para o serviço de *proxy* está ilustrado na Figura 4.13 e as principais classes definidas são:

- *ProxyService*: esta é a classe principal do serviço de *proxy* e armazena a árvore de recursos, que contém os agentes de recurso associados aos serviços disponibilizados. Disponibiliza métodos para registro, remoção e busca de agentes e serviços. Esta classe inicia uma instância da classe *CheckAvailableAgents* que verifica os agentes que possuem tempo de permanência expirado;

- *CheckAvailableAgents*: esta classe é responsável por verificar quais agentes registrados no serviço de *proxy* possuem o tempo de permanência expirado e removê-los da árvore de recursos;

- *ProxyAgentBean*: esta classe representa um agente de recurso no serviço de *proxy*. Os atributos presentes são uma URI, que se refere ao endereço do agente, uma lista de serviços providos pelo agente, uma descrição e o tempo de permanência no serviço de *proxy*;

- *RedirectApplication*: esta classe se refere a aplicação que irá receber as requisições HTTP e associá-las a alguma ação. Neste caso, a aplicação vai incluir um filtro para redirecionamento dos serviços para os agentes (*ProxyFilter*) e também classes responsáveis pela inserção, listagem e remoção de serviços e agentes (*ProxyAgentResources*, *ProxyServicesResource*);

- *ProxyAgentResources*: esta classe é responsável por invocar as operações de registro de agentes presentes na classe *ProxyService* e também retornar uma lista de agentes ativos com infor-

mações sobre eles. Se relaciona com a classe *ProxyServiceParser*, pois invoca métodos para efetuar o processamento da resposta na representação adequada ao cliente;

- *ProxyServicesResource*: esta classe é responsável por retornar uma lista dos serviços ativos no serviço de *proxy*, o que inclui todos os serviços disponilizados pelos agentes. Difere da classe *ProxyAgentResources* no sentido que ela apresenta apenas os serviços mas não efetua associação com os agentes responsáveis. Se relaciona com a classe *ProxyServiceParser*, no sentido que invoca métodos para efetuar o processamento da resposta na representação adequada ao cliente;

- *ProxyFilter*: esta classe é um filtro responsável por interceptar requisições a serviços disponibilizados e redirecioná-las ao agente responsável por esse serviço. Deve retornar ao cliente a resposta que foi recebida do agente;

- *ProxyServiceParser*: esta classe é responsável por processar os dados referentes aos agentes da árvore de recursos do serviço de *proxy* e disponibilizá-la pra no formato XML ou HTML, conforme a representação requisitada pelo cliente;

- *NodeMap*: esta classe é a estrutura de árvore de recursos presente no serviço de *proxy*, apresentando métodos para busca, inserção e remoção de elementos;

- *Node*: classe que representa um elemento, ou nó, da estrutura de árvore de recursos referente a classe *NodeMap*.

4.1.3 Diagramas de Sequência

Os diagramas de sequência são utilizados para representar as interações entre objetos ao longo do tempo através de mensagens. É possível verificar como os objetos se comunicam e quais mensagens, ou operações, são invocadas. Na modelagem da plataforma MIS estes diagramas são utilizados para ilustrar as interações entre os objetos e entre os clientes e serviços.

Os diagramas de sequência serão separados pelas principais operações na utilização dos serviços da plataforma MIS, visando uma melhor representação das funcionalidades e comunicação entre classes e objetos.

Diagramas de sequência para o serviço de notificação

A Figura 4.14 apresenta a sequência de mensagens e o relacionamento entre classes e elementos quando um evento é enviado para o serviço de notificação. Pode-se observar também o envio de notificação aos clientes subscritos. De maneira geral, um produtor de eventos envia uma mensagem, por uma requisição HTTP POST ao serviço de notificação, onde são executados três passos básicos. Primeiro é verificado se há um cliente subscrito que tenha interesse neste evento, e então notifica-se este cliente. Depois o evento é armazenado em memória no serviço e em seguida é verificado se este evento deve ser enviado ao serviço de *logging*. Em caso afirmativo, o evento será encaminhado a este serviço.

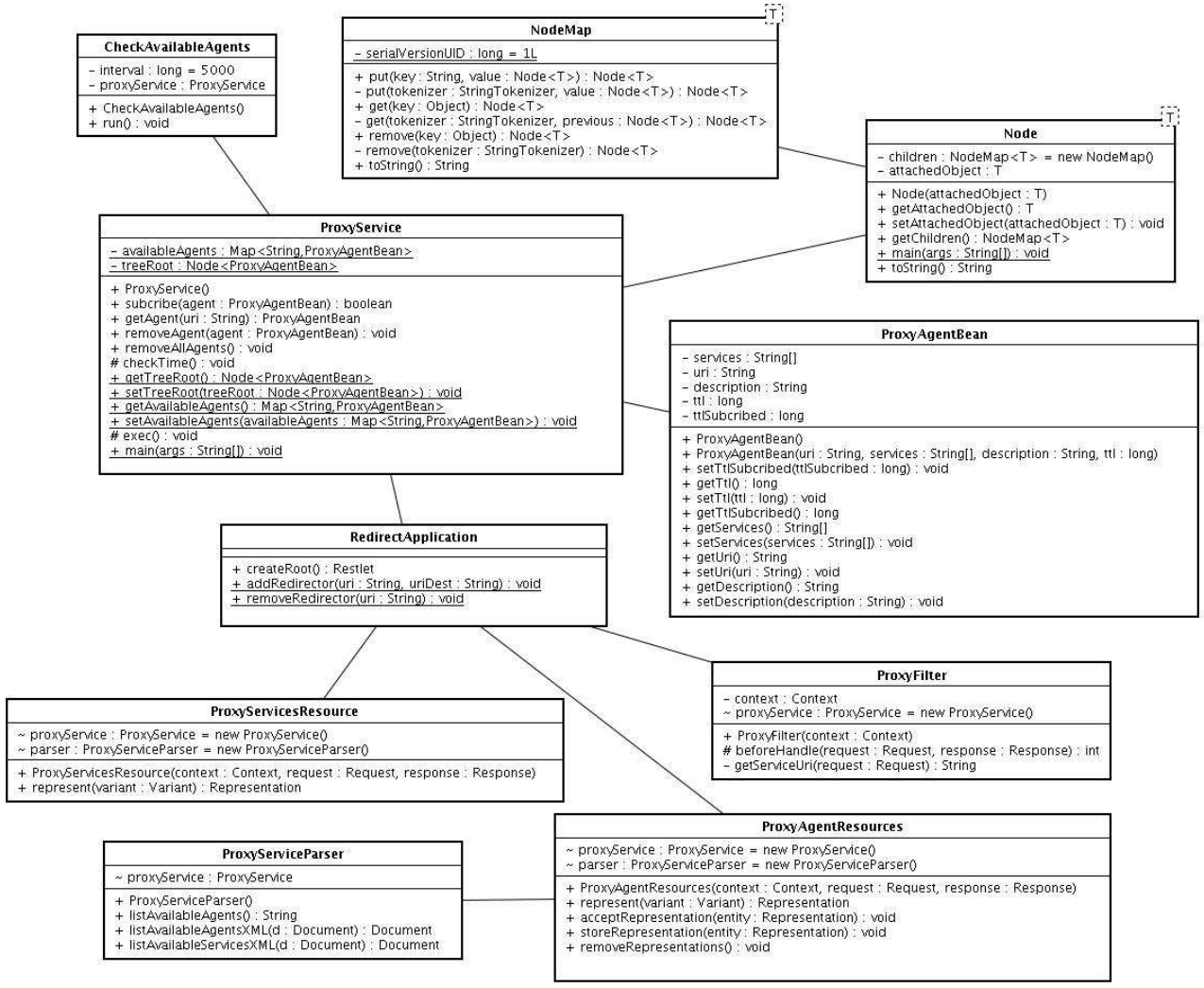

Fig. 4.13: Classes do serviço de *proxy*.

O registro de um cliente no modo *push* do serviço de notificação pode ser observado no diagrama da Figura 4.15. Neste caso um cliente envia uma requisição HTTP POST ao serviço de notificação, que irá adicionar o cliente com seus dados em uma lista em memória.

O diagrama de sequência para se efetuar uma invocação *pull* no serviço de notificação pode ser observado na Figura 4.16. Um cliente envia uma requisição HTTP GET ao serviço de notificação, que seleciona os eventos de acordo com o parâmetro XPath passado. Em seguida, efetua o processamento da resposta no formato XML devolvendo-a ao cliente requisitante.

Diagramas de sequência para o serviço de Logging

O diagrama de sequência do envio de um evento para o serviço de *logging* pode ser observado na Figura 4.17. Um produtor de eventos envia um evento ao serviço de notificação que encaminha o evento ao serviço de *logging*. É efetuado um processamento do evento XML que em seguida é

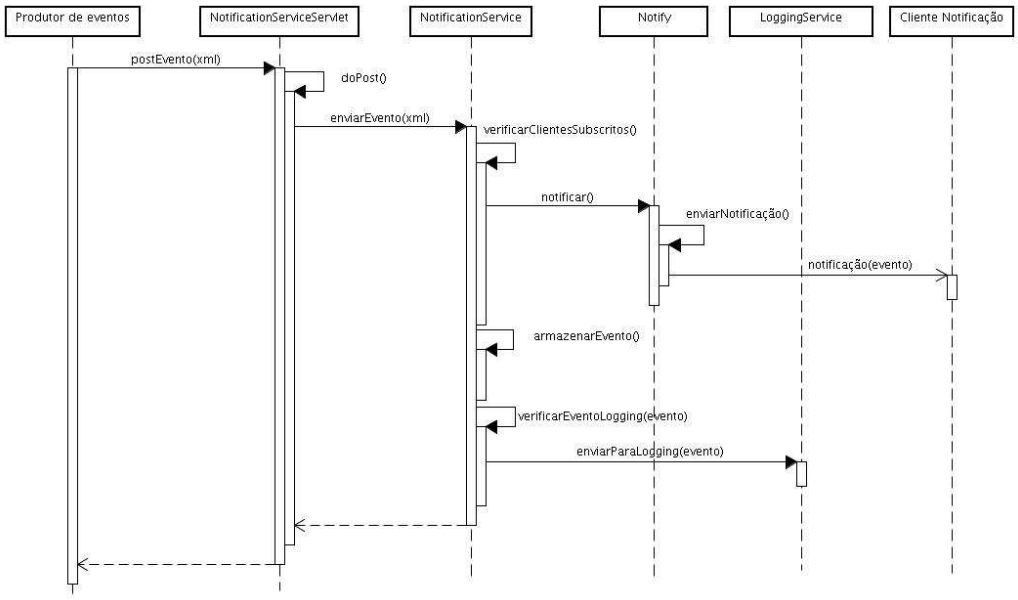

Fig. 4.14: Diagrama de sequência do envio de eventos e modo *push* no serviço de notificação.

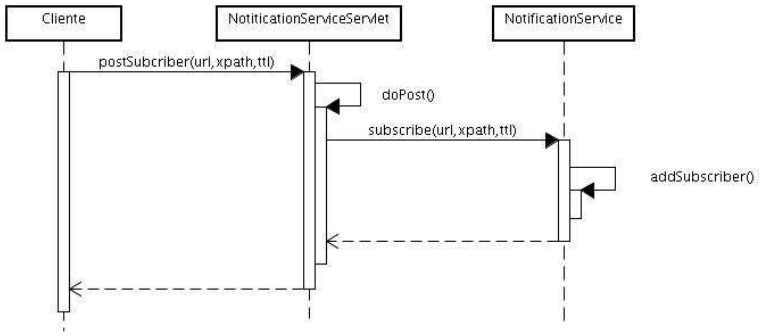

Fig. 4.15: Diagrama de sequência do registro modo *push* do serviço de notificação.

armazenado no banco de dados.

A consulta ao serviço de *logging* pode ser observada no diagrama de sequência da Figura 4.18. Um cliente faz uma requisição de consulta passando um documento XML que representa os parâmetros da consulta. O serviço de *logging* necessita fazer o processamento deste documento XML convertendo os dados em um formato que seja utilizado na consulta ao banco de dados. É efetuada a consulta no banco de dados, e, caso exista um filtro XPath, ele será aplicado no resultado da busca. Em seguida, a resposta é convertida para o formato XML e retornada ao cliente.

Diagramas de sequência para o serviço de relatório

O diagrama de sequência para a geração de um relatório pode ser observado na Figura 4.19. O cliente faz uma requisição HTTP GET passando um tipo de relatório e, opcionalmente, possíveis parâmetros. Primeiramente é buscado no banco de dados o tipo do relatório. Em seguida é gerado

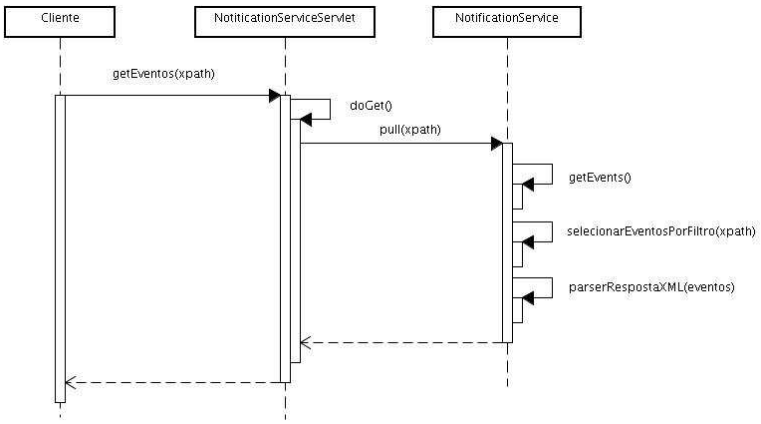

Fig. 4.16: Diagrama de sequência do modo *pull* no serviço de notificação.

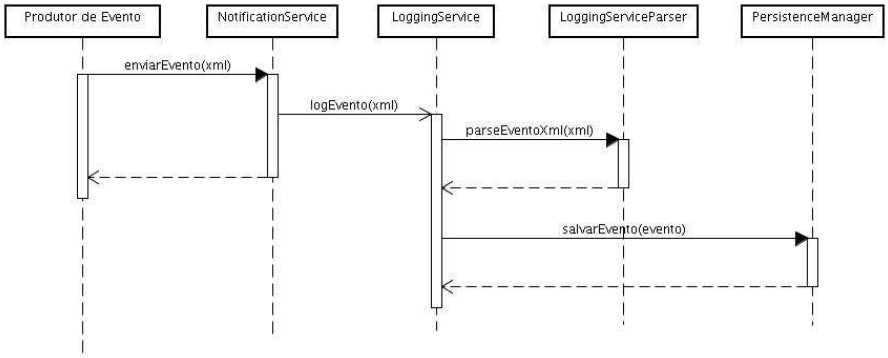

Fig. 4.17: Diagrama de sequência do envio de evento para o serviço de *logging*.

uma requisição XML baseada nos dados do tipo de relatório com os parâmetros para busca no serviço de *logging*. Posteriormente, é consultado o serviço de *logging* com essa requisição. De posse da resposta obtida é feito um processamento de formatação para gerar a resposta ao cliente.

O cadastro, alteração e listagem de tipos de relatório seguem um modelo de sequência semelhantes e podem ser observados na Figura 4.20. O cliente envia a requisição desejada ao serviço de relatório, que deverá efetuar operações para consultar ou inserir dados do tipo de relatório no banco de dados.

Diagramas de sequência para serviço de *proxy*

No serviço de *proxy* as requisições são feitas aos serviços providos pelos agentes de recurso que estão ativos. O diagrama de sequência da Figura 4.21 ilustra este processo. O serviço de *proxy* captura a requisição e passa para um filtro de redirecionamento. Esse filtro verifica para qual serviço esta requisição se destina e efetua uma busca na árvore de recursos do serviço de *proxy* para obter o agente responsável pelo serviço requisitado. Em seguida, a requisição é redirecionada para este agente. A reposta recebida do agente é encaminhada para o cliente.

O registro de agentes de recurso no serviço de *proxy* é feito pelo agente de descoberta. O diagrama de sequência deste processo pode ser observado na Figura 4.22. Primeiramente, um agente de recurso

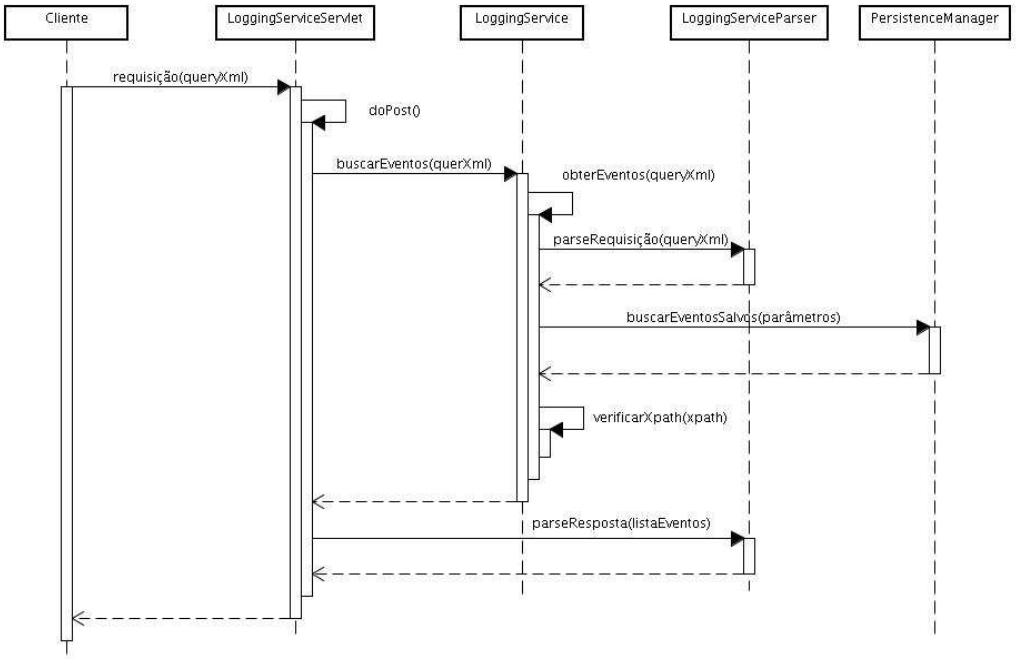

Fig. 4.18: Diagrama de sequência da consulta de eventos no serviço de *logging*.

ao ser iniciado efetua um registro no agente de descoberta. Este por sua vez verifica periodicamente quais agentes estão ativos. Os agentes de recurso que foram verificados como ativos são registrados junto ao serviço de *proxy* que insere o agente na árvore de recursos.

Diagramas de sequência para o serviço AAA

O serviço AAA é utilizado para a autenticação e autorização do cliente antes obter acesso aos serviços da plataforma MIS. O diagrama de sequência deste processo pode ser observado na Figura 4.23. O cliente ao fazer uma requisição a um serviço da plataforma MIS envia credenciais de autenticação. O serviço AAA verifica estas credenciais e, em caso positivo, libera o acesso ao serviço requisitado.

4.1.4 Diagrama de Componentes

Os diagramas de componentes ilustram em um nível mais alto de abstração, as partes que compõe o software. Um componente pode ser constituído por um conjunto de classes que formam blocos funcionais. No caso da plataforma MIS, este diagrama representa os serviços e as interfaces de comunicação entre eles e os clientes.

A Figura 4.24 exibe o diagrama de componentes da plataforma MIS onde se pode observar as interfaces de comunicação entre os serviços, agentes e clientes. Um cliente pode acessar cada serviço da plataforma MIS por uma interface de comunicação e também pode prover uma interface para receber notificações caso ele se registre no serviço de notificação no modo *push*. O serviço de notificação e relatório comunicam-se com o serviço de *logging* pelas interfaces providas por este serviço. No serviço de *proxy*, há além da interface de comunicação com o cliente, uma interface para comunicação

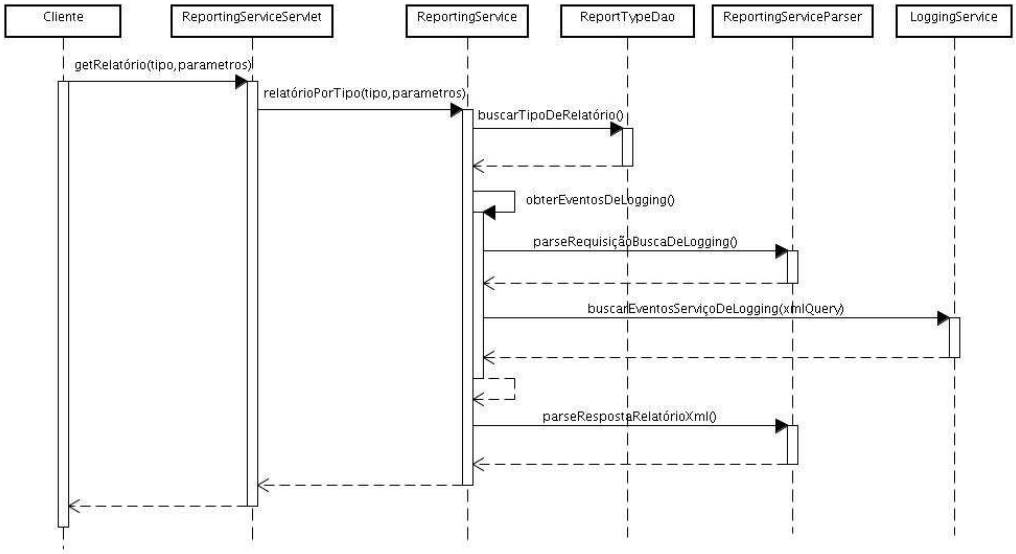

Fig. 4.19: Diagrama de sequência da geração de relatório.

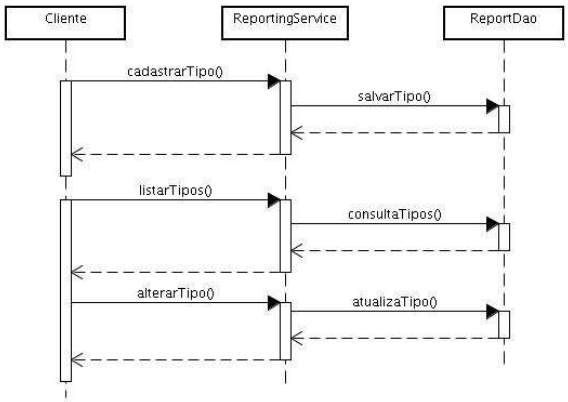

Fig. 4.20: Diagrama de sequência para o tipo de relatório.

com o agente de descoberta que pode efetuar o registro de agentes de recurso. O agente de descoberta provê uma interface para que agentes de recurso possam se registrar ao serem iniciados. Os agentes de recurso possuem interface para comunicação com o serviço de *proxy* e o agente de descoberta.

4.2 Implementação da plataforma MIS

A implementação da plataforma MIS foi baseada na modelagem proposta na Seção 4.1. Foi aplicado o padrão de projeto *Model View Controller* (MVC) [41] para a implementação. O desenvolvimento utiliza a plataforma Java EE [42] que foi escolhida por implementar os conceitos MVC multicamadas. Java EE é baseada em componentes que são executados em um servidor de aplicações.

Toda comunicação com as aplicações, serviços e componentes é feita através do protocolo HTTP. Desta forma os serviços fazem uso de Servlets Java [43] para interação das requisições HTTP com a

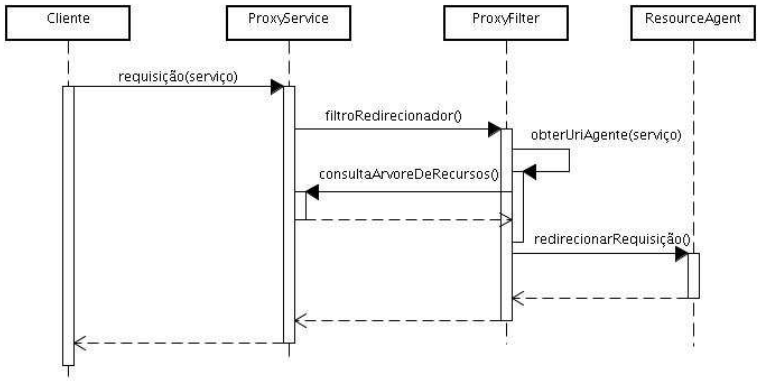

Fig. 4.21: Diagrama de sequência do serviço de *proxy*.

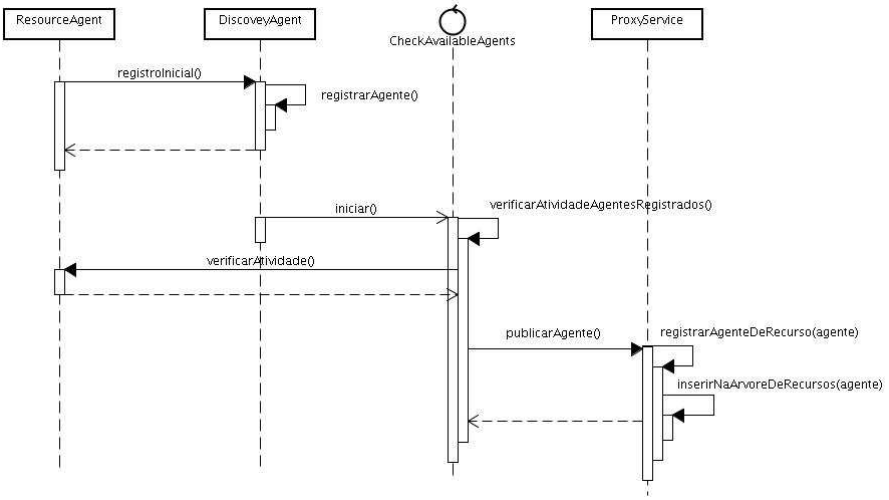

Fig. 4.22: Diagrama de sequência do registro de agentes de recurso e agente de descoberta.

aplicação que é executada no servidor. O servidor de aplicações utilizado para desenvolvimento foi o Apache Tomcat [44] em sua versão 6.

Na plataforma MIS as informações são trafegadas na forma de documentos XML entre os serviços e clientes. Para o processamento destes documentos foi utilizado a biblioteca *Java Architecture for XML Binding* [45] (JAXB), que permite um mapeamento de documentos XML em objetos Java por meio de um conjunto de classes e interfaces que representam o documento e seu conteúdo. Para cada estrutura de documentos XML foi construído um XML *Schema*[1] usado na construção das classes e interfaces referentes aos tipos de documentos XML utilizados na plataforma MIS. JAXB também efetua uma validação dos documentos que são processados sendo possível gerar exceções caso um documento enviado por um cliente não tenha um formato adequado. Os detalhes referentes a implementação de cada serviço são fornecidos a seguir.

[1]Linguagem para definição de regras de validação para documentos XML, define a estrutura, conteúdo e semântica de documentos XML.

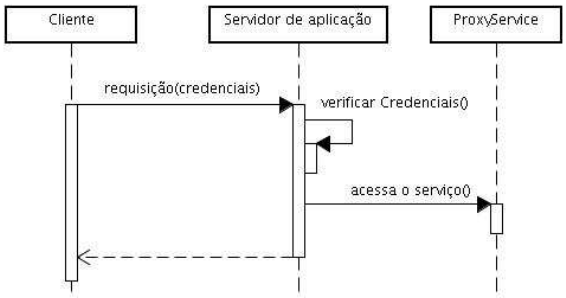

Fig. 4.23: Diagrama de sequência para o serviço AAA.

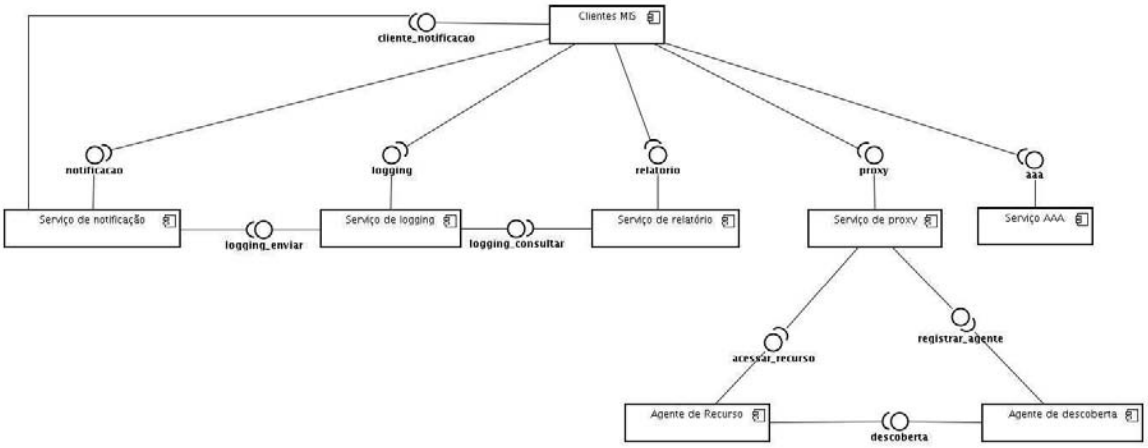

Fig. 4.24: Diagrama de componentes da plataforma MIS.

4.2.1 Implementação dos serviços

No serviço de notificação foi utilizado uma lista de documentos em memória, esta lista armazena os documentos para o modo *pull* do serviço e é compartilhada por todas as instâncias do serviço. Há também uma tabela hash que armazena os clientes subscritos no modo *push*, utilizando a URI do cliente como chave. São utilizadas threads para verificar quais eventos tiveram tempo de permanência expirado e também para efetuar a notificação de clientes quando é recebido um novo evento. Desta forma, a notificação de clientes é feita em paralelo não afetando a sequência de processamento do serviço de notificação. O serviço de notificação pode ser acessado através da URI `http://[endereço IP do servidor]:[porta]/mis/notification/`.

No serviço de *logging* a tecnologia utilizada para persistência de dados foi Java Persistence API [46] (JPA) com implementação Hibernate [34]. JPA permite o mapeamento de objetos Java persistentes, denominados entidades, em tabelas no banco de dados objeto-relacional. O banco de dados utilizado foi o PostgreSQL. Este modelo adotado permite flexibilidade quanto à base de dados utilizada e, caso necessário, é realizada a troca da base apenas com uma modificação em um arquivo de configuração do sistema. Um processamento para tradução dos eventos XML em entidades persistentes é executado neste serviço. A consulta a eventos é feita a partir de documentos XML contendo parâmetros de busca ao *logging* (exemplo no Apêndice A, Figura A.2). É executado a tradução deste

XML para uma sintaxe de busca específica do JPA. A consulta ao serviço de *logging* pode ser acessada através da URI `http://[endereço IP do servidor]:[porta]/mis/logging/`.

O serviço de relatório possui um Servlet Java para requisição de relatórios em XML. As funcionalidades de cadastro, listagem e alteração dos tipos de relatório são integradas junto a interface de gerência. Deste modo, o cliente possui uma interface Web para efetuar estas operações. Os relatórios podem ser gerados através de uma requisição na URI `http://[endereço IP do servidor]:[porta]/mis/reporting/`.

O serviço AAA é implementado no próprio servidor de aplicações Tomcat. Esse servidor possui implementação dos métodos de autenticação e autorização do protocolo HTTP. Através de configurações no servidor é possível habilitar estas funcionalidades. O cliente por sua vez deverá se autenticar através dos métodos HTTP para esta finalidade. Toda requisição para algum serviço da plataforma MIS passa por uma prévia autenticação no servidor e que, caso não se confirme, é retornado um erro. Uma outra possibilidade que foi implementada é a autenticação através de filtros[2], presentes na especificação da plataforma Java EE. Estes filtros podem dinamicamente interceptar requisições e interpretar informações contidas, como por exemplo, credenciais do usuário, permitindo com isto o fluxo normal da requisição ou então seu bloqueio retornando um erro como reposta.

O serviço de *proxy* é baseado nos conceitos da arquitetura REST. Para a implementação foi utilizado a biblioteca Restlet[3] que provê todas funcionalidades e conceitos do REST. O nome das interfaces e classes da API do Restlet derivaram diretamente dos conceitos do REST, o que facilita a implementação e uso dos conceitos da forma correta dessa arquitetura. Os agentes de recurso e agente de descoberta também fazem uso dessa biblioteca. Para cada ação desejada em um recurso é implementado algum método, no caso do HTTP os métodos POST, GET, PUT e DELETE. No serviço de *proxy* é utilizado um conceito de filtros e redirecionadores da biblioteca Restlet, que interceptam uma requisição e a encaminham para o respectivo agente de recurso. O serviço de *proxy* pode ser acessado através da URI `http://[endereço IP do servidor]:[porta]/mis/proxy`.

4.2.2 Interface de gerência

A implementação da interface de gerência da plataforma MIS é baseada no modelo Web. Desta forma, é possível ser acessada por um navegador Web em qualquer lugar que tenha acesso à rede onde está sendo executada a plataforma. Esta interface também segue o padrão Java EE e possui acesso direto aos serviços disponibilizados pela plataforma MIS. A interface de gerência é executada no servidor de aplicação Tomcat. Deste modo, é possível o controle sobre todos os dados e operações implementados nos serviços.

A interface de gerência foi desenvolvida utilizando JavaServer Faces [47] (JSF). Esta API permite uma comunicação de elementos HTML da camada de visão com os elementos da camada de modelo e, também, implementar ações de controle e fluxo de dados. O *framework* Richfaces foi utilizado para implementação de funcionalidades AJAX (Asynchronous Javascript And XML) e efeitos visuais. Este *framework* é utilizado junto ao JSF.

A interface de gerência da plataforma MIS pode ser acessada através da URI `http://[endereço IP do servidor]:[porta]/mis`.

[2] http://java.sun.com/products/servlet/Filters.html .

[3] http://www.restlet.org .

A Figura 4.25 é uma captura da tela de visualização de eventos no serviço de notificação. Pode-se observar que a interface de gerência possui um menu superior para o acesso aos serviços providos pela plataforma MIS. Cada item deste menu permite a seleção de um serviço que possui subitens representando ações de gerência para o serviço selecionado. No caso da Figura 4.25 foi acessada a funcionalidade de visualizar todos os clientes subscritos no serviço de notificação no modo *push*. Pode-se observar detalhes referentes a cada cliente, como a URI, o filtro XPath para seleção de eventos e uma opção para remover o cliente do serviço.

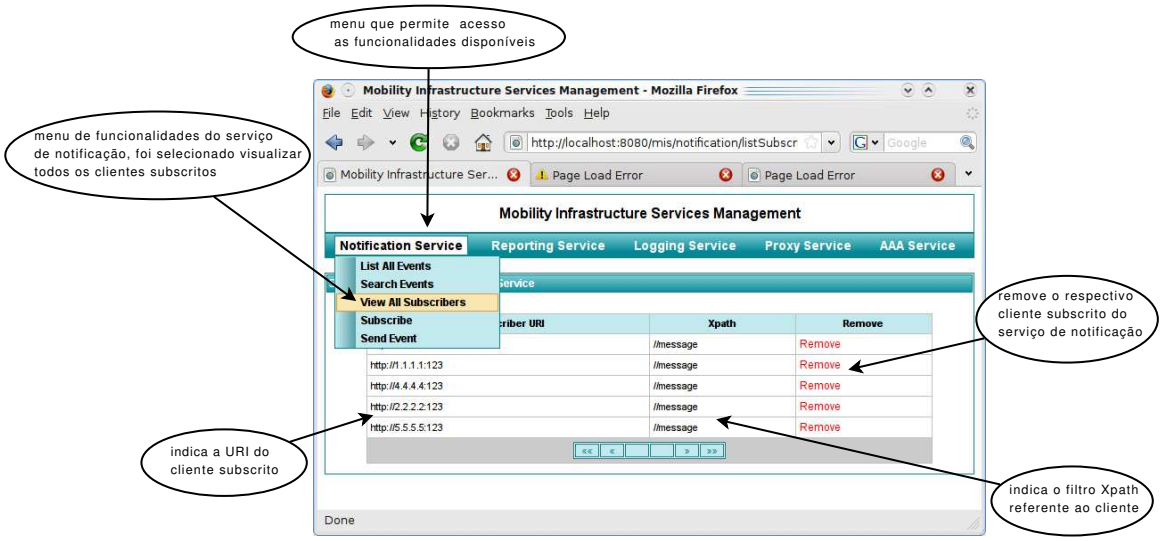

Fig. 4.25: Visualização de clientes subscritos no serviço de notificação através da interface de gerência da plataforma MIS.

A Figura 4.26 é uma captura da tela de cadastro de um tipo de relatório utilizando a interface de gerência. Nesta figura podemos observar que foi acessado o item para registro de um tipo de relatório. Na tela podem ser identificados campos com informações que podem ser inseridas ou selecionadas, referentes ao tipo de relatório que está sendo criado, como o nome, o tipo de evento, tags presente no evento, etc.

A Figura 4.27 é uma captura da tela de visualização de agentes de recurso ativos no serviço de *proxy* utilizando a interface de gerência. É possível verificar uma lista dos agentes com respectivas informações, como a URI, descrição e os serviços providos pelo agente. Há uma opção para remover o agente do serviço de *proxy*.

4.3 Testes da plataforma MIS

Os testes da plataforma MIS foram conduzidos para cada serviço provido. Para execução dos testes foram utilizados a interface de gerência e alguns agentes de testes para os serviços. Em cada serviço procurou-se comprovar e validar todas as funcionalidades propostas.

No serviço de notificação foi testado o envio de alguns eventos distintos e, em seguida, foram efetuadas requisições *pull* com diversos filtros XPath para seleção. Foram recebidas corretamente as requisições feitas. Para testar o modo *push* foi implementado um agente de teste que recebe

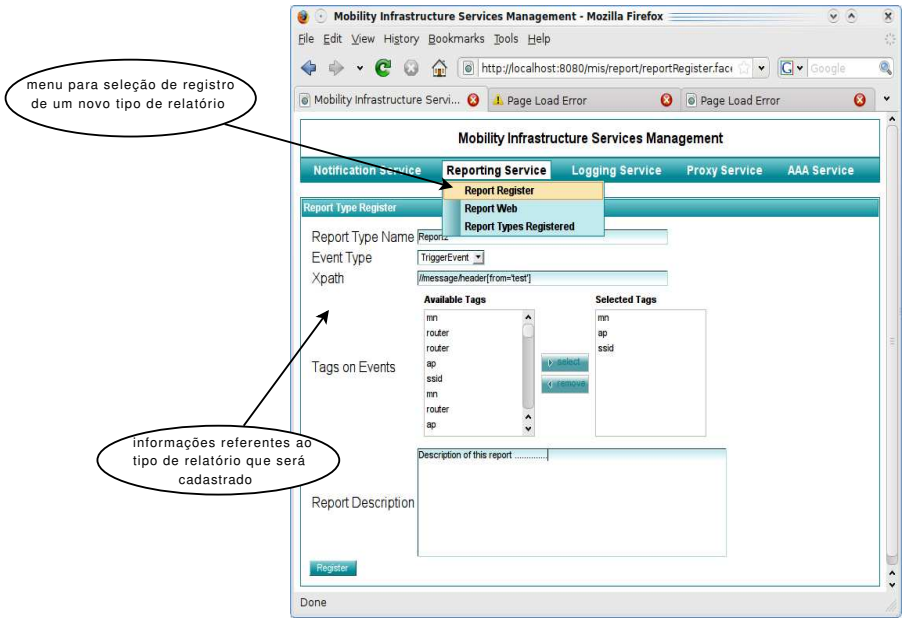

Fig. 4.26: Cadastramento de relatório usando a interface de gerência da plataforma MIS.

eventos em uma determinada porta. Foi efetuado a subscrição deste agente no serviço de notificação no respectivo endereço e porta com um filtro XPath. Em seguida, foi enviado um evento que fosse condizente ao filtro cadastrado para o agente. O agente de teste recebeu corretamente o evento que foi enviado ao serviço de notificação. De maneira similar, foi enviado um evento que não era condizente ao filtro XPath e o agente não recebeu este evento. Através da interface de gerência pôde-se verificar os dados enviados ao serviço além de facilitar no envio de eventos.

Ainda no serviço de notificação foi testada a remoção de eventos por tempo de permanência expirado. Neste caso foram inseridos diversos eventos com tempo de permanência variados. Foi comprovado que os eventos foram removidos corretamente quando o tempo de permanência se expirou.

No serviço de *logging* para testar o armazenamento de eventos, foram enviados diversos eventos marcados com a marca "log" tendo valor *true* ao serviço de notificação. Em seguida, foi verificado o banco de dados onde pôde-se comprovar que os eventos foram armazenados corretamente. Para testar a consulta de eventos foi utilizado a interface de gerência acessando o menu para consulta ao serviço de *logging* onde foram passados alguns parâmetros para busca de eventos e foram obtidos os dados esperados condizentes aos eventos enviados.

No serviço de relatório os testes foram realizados primeiramente cadastrando alguns tipos de relatório para testes com parâmetros de consulta. Em seguida foi verificado no banco de dados se os dados referentes ao tipo de relatório cadastrado foram armazenados de forma correta. Para testar os relatórios, foram efetuados requisições referentes aos tipos cadastrados. Foram obtidos dados esperados, que estavam armazenados no serviço de *logging*. A interface de gerência foi utilizada neste caso para efetuar o cadastro dos tipos de relatório, verificar os tipos cadastrados e também para requisitar um relatório no formato HTML.

No serviço de *proxy* foram feitos testes de registro de agentes e serviços. Primeiramente foram

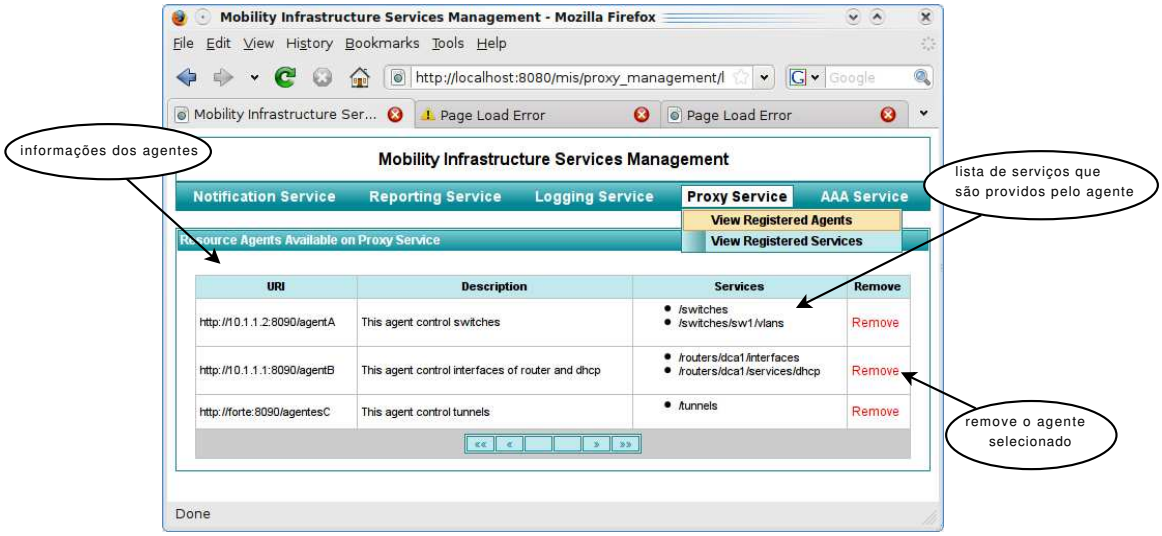

Fig. 4.27: Visualização de agentes ativos no serviço de *proxy* através da interface de gerência da plataforma MIS.

implementados agentes de recurso de testes. Estes agentes de recurso efetuam um registro inicial junto ao agente de descoberta e provêm alguns serviços de teste que permitem a visualização de informação das requisições destinadas ao serviço. É gerada uma resposta de teste para cada método HTTP. Neste teste também foi possível verificar o funcionamento do agente de descoberta que recebeu os dados dos agentes de recurso e verificou a atividade dos mesmos registrando-os junto ao serviço de *proxy*. Através da interface de gerência pôde-se listar os agentes de recurso de testes que foram cadastrados e os respectivos serviços providos por estes agentes.

Os testes de redirecionamento de requisições para os agentes de recurso no serviço de *proxy* foi feito através da implementação de um outro agente de testes HTTP. Este agente envia diversos tipos requisições HTTP (GET, POST, PUT, DELETE) à URI especificada e permite a visualização da resposta obtida. Nestes testes foram efetuadas diversas requisições aos serviços providos pelos agentes cadastrados. Verificou-se que o redirecionamento das requisições foi correto para os agentes, sendo possível visualizar nos agentes de recurso as requisições que foram feitas e no agente de testes HTTP a resposta condizente com a requisição feita ao agente.

Ainda no serviço de *proxy* foi testada a remoção de agentes por tempo de permanência, onde foi desativado um agente de recurso de teste. O agente de descoberta verificou que o agente de recurso não estava mais ativo e deste modo não efetuou o registro periódico junto ao serviço de *proxy*. Quando tentou-se acessar serviços providos por este agente foi obtido erro de serviço não encontrado.

No serviço AAA foram realizados testes de autenticação utilizando métodos HTTP para este propósito. O servidor de aplicações foi configurado para efetuar a autenticação. Foi implementado um agente de testes de autenticação, este agente acessou os serviços disponibilizados pela plataforma MIS e previamente deveria se autenticar seguindo métodos HTTP de autenticação. Quando utilizando credenciais válidas foi possível acessar os serviços normalmente, contudo, ao se utilizar credenciais inválidas foi retornado erro de autenticação conforme esperado.

4.3.1 Testes de desempenho da plataforma MIS

Foram conduzidos testes visando medir o desempenho da plataforma MIS por meio do tempo de resposta de alguns serviços. Estes testes foram executados em uma máquina com processador Intel Core Duo T2300 de 1.66GHz com 3GB de memória. A plataforma MIS e os agentes de testes foram ambos executados na mesma máquina, com isso o atraso imposto pela rede é reduzido permitindo uma melhor verificação do desempenho dos serviços da plataforma.

Primeiramente no serviço de notificação foram submetidos 100 eventos. Em seguida, foi feito uma requisição *pull* passando um filtro XPath que retornasse um único evento. A resposta a esta requisição foi obtida na média em 350 ms. Em um outro teste foram submetidos 1.000 eventos e uma requisição similar retornando um único evento. Neste caso, a resposta foi obtida na média em 1.500 ms.

Prosseguindo os testes de desempenho foram enviados 10.000 eventos com a marca "log" com valor *true*, para serem armazenado no serviço de *logging*. Em seguida, foi enviada uma requisição ao serviço de *logging* passando parâmetros para que fosse retornado um único elemento do banco de dados (desta forma, o filtro XPath é aplicado apenas uma vez no elemento retornado). A resposta a essa requisição foi recebida em 200 ms na média. Posteriormente, foi feito uma requisição de modo a retornar 100 eventos pré selecionados no banco de dados e, em seguida, aplicado um filtro XPath no resultado obtido para que fosse retornado um único evento. O tempo de resposta para esse teste foi de aproximadamente 500 ms na média. Os tempos obtidos se diferem ao do testes executados no banco de dados da Seção 3.3.1, pois a consulta utilizou parâmetros de busca diferentes e a máquina de teste apresentar melhor desempenho.

Analisando esses testes de desempenho nos serviços de notificação e *logging* é possível concluir que:

- Filtros XPath aplicados em documentos XML são relativamente custosos, demandando elevado tempo de processamento;
- O serviço de notificação tem seu desempenho no modo *pull* prejudicado caso o tempo de permanência de eventos seja elevado, fato este que ocasiona acúmulo elevado de eventos, diminuindo com isto a performance de consulta;
- No serviço de *logging* a consulta ao banco de dados é eficiente. Quanto melhor forem os parâmetros de busca escolhidos, menor o resultado da pré seleção. Com isso há um desempenho mais elevado no serviço, já que a consulta ao banco de dados é relativamente rápida, mas os filtros aplicados ao resultado são custosos.

Para testar o desempenho do serviço de *proxy* ao redirecionar requisições, foram conduzidos testes com requisições feitas ao serviço de *proxy* destinadas a um agente de recurso. Em seguida, foram realizadas as mesmas requisições, porém, destinadas diretamente ao agente de recurso, sem que tivesse a necessidade de passar pelo serviço de *proxy*. Nos testes realizados pôde-se observar que na média a diferença de resposta das requisições feitas no serviço de *proxy* e diretamente ao agente foi de aproximadamente 100 ms. Com esses resultados conclui-se que o serviço de proxy adiciona às requisições um atraso dessa ordem de grandeza.

Capítulo 5

Estudos de casos de utilização da plataforma MIS

Para validar as funcionalidades da plataforma MIS proposta neste trabalho, serão apresentados dois estudos de casos que fazem uso dessa plataforma. O primeiro trata-se de uma arquitetura de *handover* suave, e o segundo uma arquitetura de agentes inteligentes para engenharia de tráfego. Através destes estudos de casos é possível verificar como a plataforma MIS pode ser utilizada bem como o suporte no desenvolvimento de aplicações para arquiteturas de mobilidade.

5.1 Uma arquitetura de *handover* suave

Foram conduzidos estudos visando melhorias de desempenho de *handover* na arquitetura MPA e com isso surgiu a proposta de uma arquitetura de *handover* suave baseada em previsão de movimento. Este é um estudo de caso na utilização da plataforma MIS. De maneira geral, quando ocorre um *handover* é enviada uma mensagem à uma aplicação de gerência de mobilidade. Essa aplicação acessa dados de *logging* e por meio desses toma a decisão de acionar elementos responsáveis por antecipar ações de *handover* visando diminuir a perda de pacotes durante esse processo.

5.1.1 Elementos da arquitetura

A arquitetura proposta pode ser observada na Figura 5.1. É possível fazer uma divisão em camadas referente a localização e funcionalidades dos componentes nessa arquitetura. Na camada de aplicações para gerência de mobilidade encontra-se o serviço de predição de *handover*, que é responsável por prever o movimento de um nó móvel e solicitar ações para determinados roteadores de acesso antecipadamente ao *handover*. Na camada MIS encontram-se os serviços providos pela plataforma MIS e, também, agentes de recurso que atuam junto ao serviço de *proxy*. Neste caso temos o *Handover Helper* e o *AgBicasting*. O primeiro é responsável por efetuar ações que auxiliam o *handover*, como por exemplo, autenticação antecipada. O segundo é responsável por efetuar *bicasting* de pacotes, ou seja, replicação de pacotes para o novo roteador de acesso. *Bicasting* é importante para melhorar o desempenho no processo de *handover* pois pacotes destinados ao nó móvel serão entregues tanto no roteador de acesso antigo quanto no novo, evitando com isto a perda de pacotes

enquanto se completa o *handover*. Na camada MPA encontram-se os elementos que compõe a arquitetura MPA, conforme detalhado na Seção 2.3.4. Adicionalmente, há um componente especial denominado *AgTrigger*, responsável por notificar a ocorrência de um *handover*.

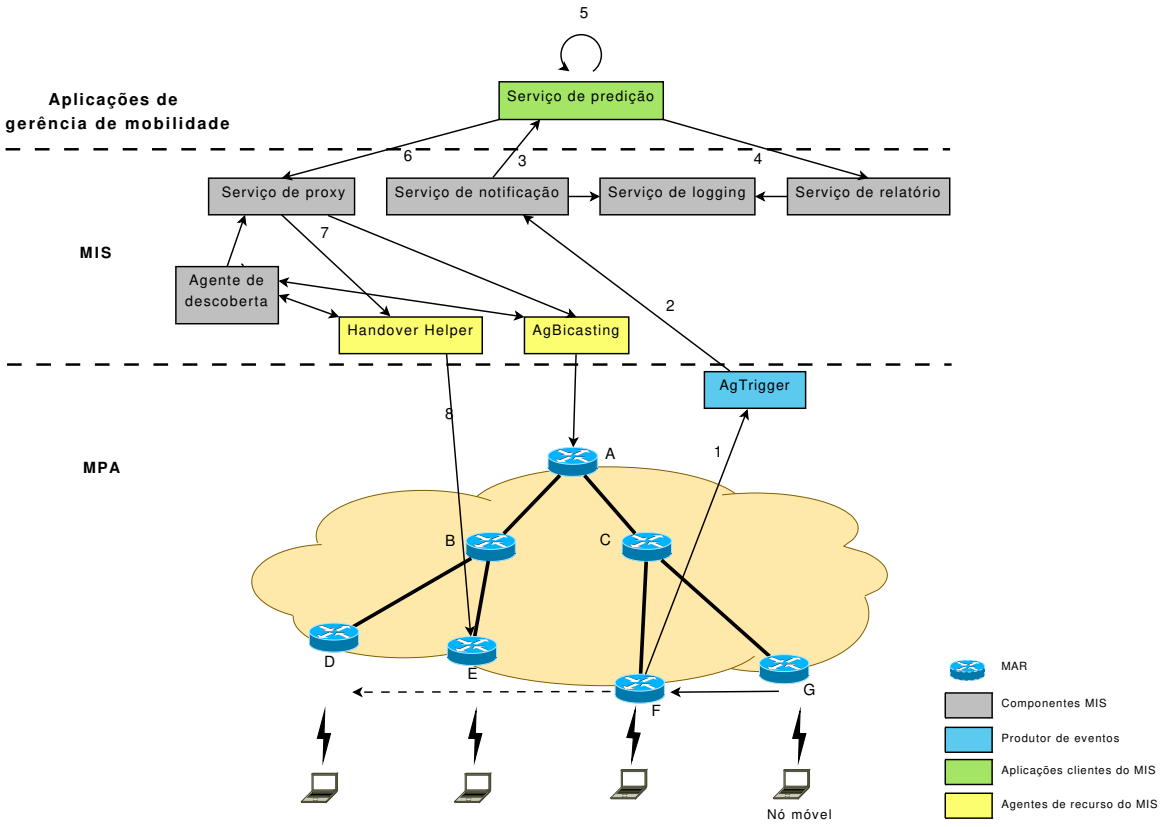

Fig. 5.1: Cenário da arquitetura de *handover* suave.

5.1.2 Funcionamento da arquitetura

Na Figura 5.1 pode-se identificar mensagens numeradas de maneira sequencial, que se referem ao fluxos de controle quando um nó móvel efetua um *handover* entre roteadores MAR de acesso. Primeiramente um nó móvel efetua um *handover* entre os roteadores MAR de acesso G e F (indicado pela seta contínua entre esses roteadores). No momento do *handover* o componente *AgTrigger* captura uma mensagem (1) que indica que o nó móvel se conectou a um novo roteador MAR de acesso e envia um evento do tipo *trigger* de associação (2) ao serviço de notificação. O serviço de predição de *handover* já registrado previamente para receber eventos do tipo *trigger* de associação receberá este evento (3) verificando que o nó móvel efetuou um *handover*. Desta maneira ele faz uma requisição (4) ao serviço de relatório, recebendo um histórico de eventos de *handover* referentes ao nó móvel. De posse desse histórico o serviço de predição de *handover* efetua um processamento (5) para determinar um provável deslocamento futuro do nó móvel. Em seguida, o serviço de predição de *handover* acessa (6) no serviço de *proxy*, os agentes de recurso *Handover Helper* e *AgBicasting*, que provêem serviços para antecipação de *handover*. O serviço de *proxy* por sua vez redireciona essas requisições

(7) para os agentes *Handover Helper* e *AgBicasting*. Os agentes por sua vez atuam diretamente nos recursos (8) da MPA. O *Handover Helper* provê ações de auxílio ao *handover* no roteador MAR de acesso E e o *AgBicasting* é responsável pela replicação de pacotes destinados ao nó móvel para os roteadores MAR de acesso E e F, agindo no roteador MAR A que possui conexão com os ramos que ligam esses dois roteadores MAR.

As informações presentes no serviço de *logging* referentes ao histórico de conexões do nó móvel foram armazenadas por meio de eventos do tipo *trigger* de associação marcados como persistentes, enviados ao serviço de notificação previamente no decorrer do tempo. Esses eventos são enviados da mesma forma expressa na mensagem (2) na Figura 5.1.

Para testar a integração da plataforma MIS na arquitetura proposta, foram implementados protótipos do serviço de predição de *handover* e dos agentes de recurso, *AgBicasting* e *Handover Helper*. Foram conduzidos testes iniciais utilizando os protótipos e a plataforma MIS implementada. Primeiramente foram enviados eventos do tipo *trigger* de associação (referentes a um nó móvel) marcados como persistentes para o serviço de notificação. Os eventos foram armazenados corretamente no serviço de *logging*. Foi criado um tipo de relatório para retornar eventos de associação referentes ao nó móvel. O serviço de predição de *handover* efetuou um registro para receber eventos do tipo *trigger* de associação e, ao ser enviado um evento deste tipo ao serviço de notificação, o serviço de predição de *handover* que havia sido registrado, recebeu a notificação corretamente. O teste no serviço de relatório prosseguiu com sucesso, ao ser enviada uma requisição de relatório do tipo previamente cadastrado para recuperação dos eventos de associação. Por fim, o serviço de predição de *handover* enviou requisições via serviço de *proxy* aos agentes de recurso *AgBicasting* e *Handover Helper*, que receberam corretamente estas requisições efetuando ações necessárias. Nos testes realizados foram obtidos resultados satisfatórios, mostrando que a plataforma atingiu os objetivos.

No funcionamento da arquitetura nesse cenário pode-se verificar que são utilizados o serviço de notificação para recebimento de eventos pelo serviço de predição de *handover*; o serviço de *logging* e relatório para o armazenamento e recuperação de eventos de conexão de nós móveis. O serviço de *proxy* é acessado para atuação na arquitetura MPA quando um *handover* é previsto. A plataforma MIS fornece suporte e facilita o desenvolvimento da arquitetura de *handover* suave. Essa arquitetura está sendo implementada como um trabalho distinto no grupo de pesquisa e será parte de uma tese de doutorado [4].

5.2 Uma arquitetura de agentes inteligentes para engenharia de tráfego

Um estudo de caso proposto para utilização da plataforma MIS é uma arquitetura de agentes inteligentes para engenharia de tráfego. Essa arquitetura consiste em agentes que podem atuar na arquitetura MPA controlando as classes de serviço (CoS) nos roteadores MAR e também na reconfiguração da topologia lógica de túneis. O objetivo destas ações é melhorar o desempenho da rede conforme a situação que se encontre o tráfego, evitando a perda e descarte de pacotes. De maneira sucinta, um agente terá como entrada informações referentes ao estado da rede e das classes de serviço de cada roteador MAR. Conforme o número de pacotes que estiverem sendo descartados e enlaces sobrecarregados, os agentes poderão atuar requisitando alteração nas configurações das classes de serviço ou da topologia lógica. O objetivo da atuação é melhorar o desempenho da rede evitando

consumo desnecessário de recursos em enlaces sobrecarregados e também minimizar o descarte de pacotes para classes de alta prioridade.

5.2.1 Elementos da arquitetura

A arquitetura de agentes inteligentes para engenharia de tráfego pode ser observada na Figura 5.2. Os elementos que compõe a arquitetura são posicionados em três camadas. A camada de agentes de engenharia de tráfego contém os agentes *AgCoS*, *AgRerouting* e *AgET*. *AgCoS* é um agente que pode atuar nas classes de serviço dos roteadores MAR conforme informações referentes aos enlaces e classes de serviço enviadas a ele. O agente *AgRerouting* tem a função de reroteamento de túneis na MPA, sendo responsável por rearranjar a topologia lógica da rede de acordo com informações referentes ao estado dos túneis. O agente *AgET* é responsável por monitorar a rede e, conforme o estado da rede, decidir acionar o agente *AgCoS* ou *AgRerouting* visando melhorar o desempenho da rede. Na camada MIS encontram-se os serviços que compõe esta plataforma e também os agentes de recurso *AgMAR* e *AgTopology*. O agente *AgMAR* é responsável por controlar recursos presentes nos roteadores MAR, como por exemplo, as classes de serviço, atuando nas filas de encaminhamento das interfaces de rede por meio da ferramenta `tc` [26]. É possível obter o estado das filas de encaminhamento instaladas em um determinado roteador MAR, definir e alterar parâmetros das filas instaladas, por exemplo, banda mínima garantida em cada fila ou banda máxima do túnel. O agente *AgTopology* tem conhecimento de quais são os roteadores MAR que compõe a rede física e também qual a topologia atual de túneis (rede lógica). Esse agente é responsável por consultar, criar e remover a topologia de túneis. No momento da criação dos túneis o agente *AgTopology* passa os parâmetros do túnel e das classes de serviço.

A camada MPA contém os elementos que constituem a arquitetura MPA conforme detalhado na Seção 2.3.4.

5.2.2 Funcionamento da arquitetura

No cenário ilustrado na Figura 5.2 o agente *AgET* primeiramente faz uma requisição para o agente *AgTopology* para descobrir quais são os roteadores MAR ativos na arquitetura MPA e qual é a topologia lógica (túneis). De posse destas informações o agente *AgET* efetua um monitoramento em todos os roteadores MAR através de *polling* no serviço de *proxy*, para saber qual o estado da rede. Através desse monitoramento são obtidas informações, como por exemplo, taxa de pacotes descartados em cada fila de encaminhamento, banda passante de cada fila, banda mínima garantida para cada fila e banda máxima do túnel. O serviço de *proxy* por sua vez encaminha estas requisições de monitoramento para o agente *AgMAR* que é o agente de recurso responsável por obter as informações dos roteadores MAR. O agente *AgET* analisa as informações obtidas visando métricas para melhoria de desempenho e controle de congestionamento. Por meio de um algoritmo de decisão ele aciona o agente *AgCoS* ou *AgRerouting* enviando um evento ao serviço de notificação. Os agentes *AgCoS* e *AgRerouting* estão previamente registrados no serviço de notificação para receber eventos destinados a eles. Dessa maneira, quando o agente *AgET* enviar eventos para acionar esses agentes de recurso, eles receberão esses eventos que contém informações (estado da rede, roteadores, enlaces) necessárias para iniciar um procedimento de atuação na rede.

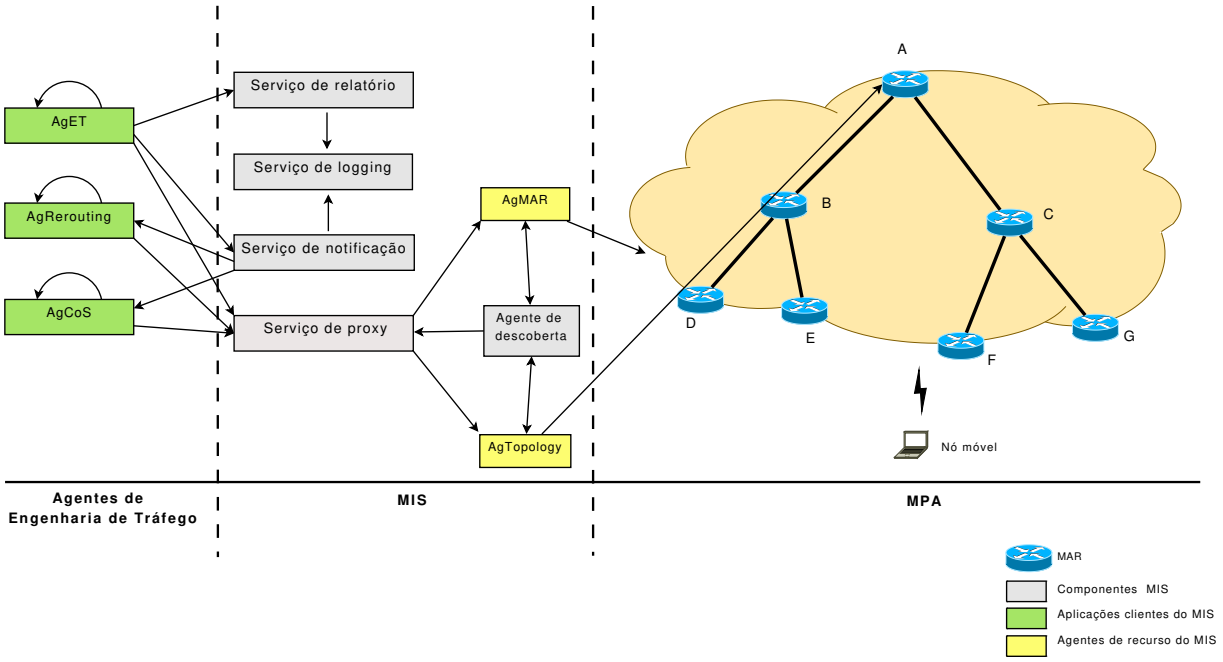

Fig. 5.2: Cenário da arquitetura de agentes inteligentes para engenharia de tráfego.

O agente *AgCoS*, ao ser requisitado, recebe informações sobre a topologia e as filas de encaminhamento, de modo que com estes dados é feito o processamento da nova configuração para essas filas de encaminhamento. O agente *AgCoS* pode ser requisitado quando o descarte de uma fila atinge um determinado nível ou quando deixa de haver descarte de pacotes nas filas. O agente recalcula os parâmetros das filas de encaminhamento dos roteadores MAR envolvidos e, via serviço *proxy*, solicita ao agente *AgMAR* a reconfiguração desses parâmetros.

Quando é solicitado uma atuação do agente *AgRerouting*, são enviados dados referentes à topologia atual, o estado dos enlaces e dos roteadores MAR. O agente *AgRerouting* efetua um processamento para o reroteamento dos túneis em uma nova topologia. Deste modo, é solicitado ao agente *AgTopology* a execução dessa nova topologia de túneis por meio do serviço de *proxy*.

No funcionamento descrito para o cenário da arquitetura de agentes inteligentes para engenharia de tráfego, é possível verificar a utilização do serviço de notificação, que é utilizado para a comunicação entre os agentes de engenharia de tráfego. O serviço de *proxy* e os agentes de recurso são utilizados para a leitura de estado e atuação nos recursos da arquitetura MPA.

Nesse mesmo cenário ainda pode-se observar a possibilidade do agente *AgET* ser um produtor de eventos, enviando eventos ao serviço de notificação que apresentem como conteúdo informações referentes ao estado da rede, tráfego, recursos e filas de encaminhamento. Essas informações são obtidas durante o processo de monitoramento por *polling*. Os eventos podem ser marcados para armazenamento em *logging*. Com isso é possível efetuar uma requisição ao serviço de relatório para obtenção de dados visando uma análise do histórico das informações armazenadas no serviço de

logging. Por meio desta análise o agente *AgET* pode tomar decisões baseadas em padrões de tráfego encontrados. Nessa abordagem são utilizados adicionalmente os serviços de *logging* e relatório.

Alguns testes funcionais foram realizados com o objetivo de verificar a integração dos elementos da arquitetura proposta com a plataforma MIS. Foram utilizados implementações iniciais dos agentes *AgMAR*, *AgTopology*, *AgCoS* e *AgET*. Nos testes realizados o agente *AgCoS* se registrou no serviço de notificação para receber eventos de atuação nas classes de serviço. O agente *AgET* efetuou monitoramento nas filas de encaminhamento dos roteadores MAR enviando requisições ao agente *AgMAR* via serviço de *proxy*. Foram obtidas informações corretamente e, em seguida, enviado um evento de atuação nas classes de serviço ao serviço de notificação. O agente *AgCoS* que havia se registrado, recebeu corretamente o evento e solicitou a reconfiguração das classes de serviço nos roteadores MAR enviando uma requisição ao serviço de *proxy* que a encaminhou ao agente *AgMAR* corretamente. Nos testes funcionais realizados foram obtidos resultados satisfatórios, mostrando que a plataforma atingiu os objetivos propostos.

A implementação da arquitetura de agentes inteligentes para engenharia de tráfego está sendo desenvolvida como um trabalho distinto no grupo de pesquisa constituindo parte de um trabalho de mestrado [48].

Capítulo 6

Conclusões

Diversos sistemas que apresentam soluções de micromobilidade vêm sendo propostos como resposta ao crescente interesse por tecnologias de mobilidade IP. A arquitetura MPA é uma solução de micromobilidade baseada em tunelamento de pacotes destinados aos nós móveis que visitam um domínio. Na arquitetura MPA foram realizados estudos visando diminuir o tempo de *handover* e dessa maneira diminuir o número de pacotes perdidos neste processo. Melhorar o desempenho da rede lidando com situações de congestionamento de enlaces e descartes de pacotes em classes de serviço foi uma outra área de estudo nesta arquitetura. Durante a implementação da arquitetura MPA e dos estudos realizados, ficou evidente que diferentes aplicações demandavam um conjunto de funcionalidades comuns.

A plataforma MIS apresenta uma solução para o problema de comunicação entre aplicações, armazenamento persistente de informações e monitoramento e atuação em recursos da rede. A plataforma fornece suporte ao desenvolvimento de aplicações de gerência da rede, engenharia de tráfego e gerência de mobilidade. Em arquiteturas de mobilidade muitas destas funcionalidades são implementadas pelas próprias aplicações. A proposta do presente trabalho foi de centralizar e padronizar estas funcionalidades para que possam ser utilizadas por diversas aplicações que venham a surgir, bem como permitir que esta arquitetura proposta seja expandida, adicionando-se mais funcionalidades conforme sejam necessárias. A plataforma MIS funciona como uma camada intermediária no suporte a aplicações para arquiteturas de mobilidade, facilitando o desenvolvimento dessas aplicações.

Os serviços disponibilizados pela plataforma MIS foram: serviço de notificação, responsável pela difusão de eventos e informação na arquitetura; serviço de *logging*, responsável pelo armazenamento de eventos para análise futura; serviço de relatório, responsável pela recuperação de eventos armazenados; serviço de *proxy*, centraliza o acesso e controle dos recursos da rede e serviço AAA que é responsável pelo acesso seguro à plataforma MIS.

A plataforma MIS foi utilizada em uma arquitetura de agentes inteligentes para engenharia de tráfego e em uma arquitetura de *handover* suave. Através destas aplicações pôde-se comprovar como é possível integrar aplicações de um nível mais alto com a plataforma proposta utilizando as funcionalidades providas pelos serviços. Estes serviços fornecem suporte para comunicação, acesso a informações persistentes e atuação nos recursos da rede. Testes iniciais com protótipos da aplicação de predição de *handover* e dos agentes de engenharia de tráfego demonstram que a plataforma MIS facilitou o desenvolvimento destas aplicações.

O presente trabalho atingiu os objetivos propostos no sentido que a plataforma MIS forneceu

suporte efetivo às aplicações em arquiteturas de mobilidade, compartilhando funcionalidades que facilitam a implementação além de motivar o surgimento de novas aplicações.

6.1 Trabalhos Futuros

O objetivo da plataforma MIS é de reunir e compartilhar serviços que fornecem funcionalidades que possam ser de interesse comum a múltiplas aplicações. Desta maneira, a arquitetura proposta para a plataforma MIS permite que sejam adicionados novos serviços conforme necessidades que surjam durante a implementação de aplicações em arquiteturas de mobilidade. É necessário um estudo visando uma metodologia para a introdução de novos serviços na plataforma.

Uma outra área para trabalhos futuros é o estudo e desenvolvimento de aplicações em arquiteturas de mobilidade que façam uso da plataforma MIS. Além de facilitar o desenvolvimento, a plataforma também oferece novas possibilidades para atuação, motivando o surgimento de novas aplicações para arquiteturas de mobilidade.

Finalmente, o desenvolvimento de agentes de recurso de uso geral para monitoramento e atuação em arquiteturas de mobilidade é outro trabalho futuro sugerido.

Referências Bibliográficas

[1] D. Johnson, C. Perkins, and J. Arkko. Mobility support in IPv6. RFC 3775, The Internet Engineering Task Force (IETF), June 2004.

[2] Ed C. Perkins. Mobility support for IPv4. RFC 3344, The Internet Engineering Task Force (IETF), August 2002.

[3] Eduardo Zagari, Rodrigo Prado, Eleri Cardozo, Mauricio Magalhães, Tomás Badan, José Carrilho, Rossano Pinto, André Berenguel, Daniel Barboza, Daniel Moraes, Thienne Johnson, and Lars Westberg. MPA: a Network-Centric Architecture for Micro-Mobility Support in IP and MPLS Networks. In *IEEE Sixth Annual Conference on Communication Networks and Services Research - CNSR'08*, Halifax, Canada, 2008.

[4] R. Prado, E. Zagari, T. Badan, E. Cardozo, M. Magalhaes, J. Carrilho, R. Pinto, A. Berenguel, D. Barboza, D. Moraes, T. Johnson, and L. Westberg. A Reference Architecture for Micro-mobility Support in IP Networks. *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pages 624–630, July 2008.

[5] Philip A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39(2):86–98, 1996.

[6] Andrew Myles and David Skellern. Comparing four IP based mobile host protocols. *Comput. Netw. ISDN Syst.*, 26(3):349–355, 1993.

[7] A. G. Valko. Cellular IP: A New Approach to Internet Host Mobility. In *ACM Comp. Commun. Rev.*, January 1999.

[8] Charles E. Perkins. Mobile IP. *IEEE Communications Magazine*, 40(5):66–82, May 2002.

[9] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423, Internet Engineering Task Force, May 2006.

[10] Mohammed Hadi Habaebi. Macro/Micro-Mobility Fast Handover in Hierarchical Mobile IPv6. *Computer Communications*, 29(5):611–617, 2006.

[11] Hesham Soliman. *Mobile IPv6 - Mobility in a wireless Internet*. Addison-Wesley, Boston, 2004.

[12] Eduardo Zagari, Rodrigo Prado, Eleri Cardozo, Mauricio Magalhães, Tomás Badan, José Carrilho, Tiago Dolphine, André Berenguel, Daniel Barboza, Thienne Johnson, and Lars Westberg.

Design and Implementation of a Network-Centric Micro-Mobility Architecture. In *IEEE Wireless Communications and Networking Conference (WCNC 2009)*, Budapest, Hungary, 2009.

[13] Pierre Reinbold and Olivier Bonaventure. A Survey of IP micro-mobility protocols. *IEEE Communications Surveys and Tutorials*, 5:40–57, 2002.

[14] Debanjan Saha, Amitava Mukherjee, Iti Saha Misra, Mohuya Chakraborty, and N. Subhash. Mobility support in IP: a survey of related protocols. *IEEE Network*, 18(6):34–40, 2004.

[15] R. Koodli. Fast Handovers for Mobile IPv6. RFC 4068, The Internet Engineering Task Force (IETF), July 2005.

[16] Martin Dunmore. Mobile IPv6 Handovers: Performance Analysis and Evaluation. *6NET Consortium, Deliverable D4.1.3*, 2005.

[17] Sangheon Pack and Yanghee Choi. Performance Analysis of Fast Handover in Mobile IPv6 Networks. volume 2775 of *Lecture Notes in Computer Science (LNCS)*, pages 679–691. Springer-Verlag, 2003.

[18] H. Soliman, C. Castelluccia, K. El Malki, and L. Bellier. Hierarchical Mobile IPv6 Mobility Management (HMIPv6). RFC 4140, The Internet Engineering Task Force (IETF), August 2005.

[19] Fabio M. Chiussi, Denis A. Khotimsky, and Santosh Krishnan. Mobility Management in Third-Generation All-IP Networks. *IEEE Communications Magazine*, pages 124–135, September 2002.

[20] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil. Proxy Mobile IPv6. RFC 5213, IETF, August 2008.

[21] Kang won Lee, Won-Kyeong Seo, Dong-Won Kum, and You-Ze Cho. Global Mobility Management Scheme with Interworking between PMIPv6 and MIPv6. In *Networking and Communications, 2008. WIMOB '08. IEEE International Conference on Wireless and Mobile Computing*, pages 153–158, Oct. 2008.

[22] Ki-Sik Kong, Wonjun Lee, Youn-Hee Han, Myung-Ki Shin, and HeungRyeol You. Mobility management for all-IP mobile networks: mobile IPv6 vs. proxy mobile IPv6. *Wireless Communications, IEEE*, 15(2):36–45, April 2008.

[23] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS. RFC 2702, The Internet Engineering Task Force (IETF), September 1999.

[24] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and Principles of Internet Traffic Engineeriing. RFC 3272, The Internet Engineering Task Force (IETF), May 2002.

[25] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475, Internet Engineering Task Force, December 1998.

[26] Bert Hubert, Remco van Mook, Martijn van Oosterhout, Paul B. Schroeder, and Jasper Spaans. Linux Advanced Routing & Traffic Control HOWTO. http://lartc.org/, 2004.

[27] Tiago Dolphine, André Berenguel, Rodrigo Prado, and Eleri Cardozo. Plataforma de serviços de infra-estrutura para gerência de mobilidade. In *XIV Workshop de Gerência e Operação de Redes e Serviços - Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos 2009*, Recife, PE, May 2009.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[29] W3C. Extensible Markup Language (XML). `http://www.w3.org/XML/`. Acesso em julho de 2009.

[30] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). RFC 1157, Internet Engineering Task Force, May 1990.

[31] James Clark and Steve DeRose. XML Path Language (XPath), 1999. `http://www.w3.org/TR/xpath`. Acesso em julho de 2009.

[32] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language, 2007. `http://www.w3.org/TR/xquery`. Acesso em julho de 2009.

[33] Sun Microsystems. The Source for Java Developers. `http://java.sun.com`. Acesso em julho de 2009.

[34] Red Hat. Hibernate. `http://www.hibernate.org/`. Acesso em julho de 2009.

[35] Sun Microsystems. JDBC API. `http://java.sun.com/products/jdbc/overview.html`. Acesso em julho de 2009.

[36] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252, Internet Engineering Task Force, January 2006.

[37] J. Postel and J. Reynolds. TELNET Protocol Specification. RFC 854, May 1983.

[38] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. dissertation, University of California, Irvine, 2000.

[39] OMG. Unified Modeling Language. `http://www.uml.org/`. Acesso em julho de 2009.

[40] Dan Pilone and Neil Pitman. *UML 2.0 in a nutshell*. O'Reilly Media, Inc., 1 edition, 6 2005.

[41] F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons Ltd, 1996.

[42] Sun Microsystems - Sun Developer Network. Java platform, enterprise edition. `http://java.sun.com/javaee/`. Acesso em julho de 2009.

[43] Sun Microsystems. Java Servlet API. http://java.sun.com/products/servlet/. Acesso em julho de 2009.

[44] Apache Software Foundation. Apache tomcat. http://tomcat.apache.org/. Acesso em julho de 2009.

[45] JAXB Reference Implementation Project. https://jaxb.dev.java.net/. Acesso em julho de 2009.

[46] Sun Microsystems - Sun Developer Network. Java Persistence API. http://java.sun.com/javaee/technologies/persistence.jsp. Acesso em julho de 2009.

[47] Sun Microsystems - Sun Developer Network. JavaServer Faces Technology. http://java.sun.com/javaee/javaserverfaces/. Acesso em julho de 2009.

[48] André Berenguel and Eleri Cardozo. Agentes Inteligentes para Controle de Engenharia de Tráfego em Arquiteturas de Micromobilidade. In *II EADCA - Segundo Encontro dos Alunos e Docentes do Departamento de Engenharia de Computação e Automação Industrial*, Campinas, SP, 2009. http://www.dca.fee.unicamp.br/portugues/pesquisa/seminarios/2009/artigos/berenguel_cardozo.pdf. Acesso em julho de 2009.

Apêndice A

Exemplos de utilização da plataforma MIS

Este apêndice apresenta alguns exemplos de utilização e de documentos na utilização dos serviços providos pela plataforma MIS.

A.1 Serviço de notificação

Para enviar um evento para o serviço de notificação basta ser submetido uma requisição HTTP POST na URI do serviço contendo como parâmetro o evento XML. Um exemplo pode ser verificado na tabela A.1.

URI	`http://localhost:8080/mis/notification`
Método	POST
Parâmetros	
xml	documento figura 3.2

Tab. A.1: Serviço de notificação: envio de evento.

Para recuperar eventos no serviço de notificação no modo *pull*, é enviado uma requisição do tipo HTTP GET para URI do serviço, contendo o filtro XPath como parâmetro. Um exemplo para retorno de eventos do tipo "TriggerEvent" pode ser observador na tabela A.2.

URI	`http://localhost:8080/mis/notification?xpath=//message/header[type='TriggerEvent']`
Método	GET

Tab. A.2: Serviço de notificação: requisição *pull*.

O documento XML com o resultado obtido com a requisição *pull* pode ser observado na figura A.1.

Para efetuar registro de um cliente no serviço de notificação no modo *push* é enviado uma requisição HTTP POST para a URI do serviço. Um exemplo pode ser observado na tabela A.3.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<messages>
    <message>
        <header>
            <type>TriggerEvent</type>
            <from>Agente</from>
            <ttl>1249503417856</ttl>
            <seq>1</seq>
            <log>true</log>
        </header>
        <body>
            <mn>
              <mac>00:13:02:de:26:ca</mac>
              <ip>192.168.1.200</ip>
              <router>192.168.1.1</router>
            </mn>
            <ap>
              <ip>192.168.1.5</ip>
              <mac>00:13:02:de:26:ca</mac>
              <ssid>teste</ssid>
            </ap>
        </body>
        </message>
        . . .
</messages>
```

Fig. A.1: Resposta XML para *pull* no serviço de notificação.

URI	`http://localhost:8080/mis/notification`
Método	POST
Parâmetros	
uri	http://192.168.1.120:321
xpath	//message/header[type='TriggerEvent']
ttl	10000

Tab. A.3: Serviço de notificação: registro de cliente modo *push*.

A.2 Serviço de *logging*

A consulta ao serviço de *logging* pode ser feita enviando uma requisição HTTP do tipo POST contendo um documento XML de consulta na URI do serviço. Um exemplo de documento XML de consulta pode ser observado na figura A.2. Os parâmetros referentes a data na plataforma MIS seguem um padrão do tipo: yyyy/mm/dd hh:MM:ss:SSS, onde "y" se refere ao ano, "m" ao mês, "d" ao dia, "h" a hora, "M" a minuto, "s" a segundo e "S" a milissegundos.

O exemplo de uma consulta ao serviço de *logging* é expresso na tabela A.4.

URI	`http://localhost:8080/mis/logging`
Método	POST
Parâmetros	**Valor**
xml	documento figura A.2

Tab. A.4: Serviço de *logging*: requisição de consulta.

A resposta obtida com a requisição de consulta ao serviço de *logging* pode ser observada no

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<message>
  <header>
    <type>LoggingQuery</type>
    <from>PredictionService</from>
  </header>
  <body>
    <dateI>2009/07/16 15:24:02:661</dateI>
    <dateF>2009/08/06 17:09:47:745</dateF>
    <macs>
          <mac>00:16:6f:2f:e9:b8</mac>
    </macs>
    <ips>
        <ip>10.20.9.100</ip>
    </ips>
    <eventType>TriggerEvent</eventType>
    <xpath>//message</xpath>
  </body>
</message>
```

Fig. A.2: Documento XML para consulta ao serviço de *logging*.

documento da figura A.3.

A.3 Serviço de relatório

A requisição de um relatório pode ser feita enviando uma requisição HTTP do tipo GET contendo o tipo do relatório e opcionalmente parâmetros como data inicial e final, na URI do serviço. Um exemplo da requisição de um relatório pode ser observada na tabela A.5.

URI	`http://localhost:8080/mis/reporting?reportType=Report1&dateI=2009/07/202012:00:00:000&dateF=2009/08/062000:00:00:000`
Método	GET

Tab. A.5: Serviço de relatório: requisição de relatório.

A.4 Serviço de *proxy*

O serviço de *proxy* pode ser acessado através da URI do serviço de *proxy* seguida do caminho do serviço provido pelo agente de recurso a ser requisitado. Um exemplo de acesso ao serviço `routers/dca01/interfaces` pode ser observado na tabela A.6.

URI	`http://localhost:8080/mis/proxy/`**`routers/dca01/interfaces`**
Método	GET, POST, PUT, DELETE

Tab. A.6: Serviço de *proxy*: requisição de um agente de recurso.

```
1
2  <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
3  <events>
4      <event>
5          <time>2009-08-06 00:32:57.356</time>
6          <message>
7              <header>
8                  <type>TriggerEvent</type>
9                  <from>00:0c:42:1b:a7:df</from>
10                 <ttl>1249529587230</ttl>
11                 <seq>1</seq>
12                 <log>true</log>
13             </header>
14             <body>
15                 <mn>
16                     <mac>00:16:6f:2f:e9:b8</mac>
17                     <ip>10.20.9.100</ip>
18                     <router>10.20.9.13</router>
19                 </mn>
20                 <ap>
21                     <ip>10.20.9.31</ip>
22                     <mac>00:0c:42:1b:a7:df</mac>
23                 </ap>
24             </body>
25         </message>
26     </event>
27     <event>
28         <time>2009-08-06 00:36:35.312</time>
29         <message>
30             <header>
31                 <type>TriggerEvent</type>
32                 <from>00:0c:42:1b:a7:df</from>
33                 <ttl>1249529805299</ttl>
34                 <seq>3</seq>
35                 <log>true</log>
36             </header>
37             <body>
38                 <mn>
39                     <mac>00:16:6f:2f:e9:b8</mac>
40                     <ip>10.20.9.100</ip>
41                     <router>10.20.9.13</router>
42                 </mn>
43                 <ap>
44                     <ip>11.20.9.31</ip>
45                     <mac>11:0c:42:1b:a7:df</mac>
46                 </ap>
47             </body>
48         </message>
49     </event>
50 </events>
```

Fig. A.3: Resposta XML para consulta ao serviço de *logging*.