UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC
PRÓ-REITORIA DE GRADUAÇÃO
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

NEGOCIAÇÃO DE ACORDOS DE NÍVEL DE SERVIÇOS EM REDES DE COMPUTADORES

FELIPE BOTTAN TEIXEIRA

FLORIANÓPOLIS
NOVEMBRO/2010

FELIPE BOTTAN TEIXEIRA

NEGOCIAÇÃO DE ACORDOS DE NÍVEL DE SERVIÇOS EM REDES DE COMPUTADORES

Monografia submetida a Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Dr. Roberto Willrich

FLORIANÓPOLIS
NOVEMBRO/2010

Monografia de graduação sob o título "*Negociação de Acordos de Nível de Serviços em Redes de Computadores*", defendida por Felipe Bottan Teixeira e aprovada em _____ de 2010, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos professores:

Prof. Dr. Roberto Willrich
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Vitório Bruno Mazzola
Universidade Federal de Santa Catarina
Membro da Banca

Prof. Dr. Roosvelter João Coelho da Costa
Universidade Federal de Santa Catarina
Membro da Banca

À minha mãe Benedita Aparecida Bottan Teixeira e ao meu Pai Kleber Teixeira Junior

AGRADECIMENTOS

Agradeço ao meu orientador Roberto Willrich e também aos colega Achilles Prudêncio e Victor Samuel pelo apoio no desenvolvimento deste trabalho e pela oportunidade de trabalharmos juntos no Laboratório de Pesquisas em Sistemas Distribuídos – LAPESD. Agradeço também ao meus pais que sempre me apoiaram e continuam me apoiando ao longo da minha vida e carreira acadêmica.

RESUMO

Serviços de comunicação com garantias de qualidade de serviço (QoS) são cada vez mais um requisito essencial para os usuários deste tipo de serviço. E quando há garantia de QoS, os clientes e os provedores de serviço de rede devem negociar um Acordo de Nível de serviço (SLA – *Service Level Agreement*). Atualmente esta negociação e feita com base em parâmetros de qualidade específicos do provedor. Este trabalho consiste na implementação de um protótipo de negociação de acordos de Nível de serviço para uma aplicação web, oferecendo uma maior flexibilidade em termos de parâmetros para especificação da QoS durante a negociação de uma sessão SIP (do inglês *Session Initiation Protocol*). Este protótipo, baseado em avanços na área de Web Semântica, fará uso de ontologias para representar a especificação dos parâmetros de qualidade e tem o objetivo de ser um protótipo formal para sistemas de QoS.

Palavras-chave: Acordo de Nível de serviço, Qualidade de serviço, Especificação de Nível de serviço, Web Semântica, Ontologia, Redes de Computadores, Provedor de serviço .

ABSTRACT

Communication services with quality of service (QoS) are often a essential requisite for the users of this type of service. And when there is guarantee of QoS, the client and the network service providers must negotiate a Service Level Agreement (SLA). Nowadays this negotiation is done based on specific quality parameters of the provider. This work consist in an implementation of a prototype of a web application for the negotiation of service level specification offering a greater flexibility in terms of parameters to specify a QoS during the negotiation of a SIP (Session Initiation Protocol) session. This application is meant to be a prototype for QoS systems, which makes use of a semantic approach to the formal specification of the quality parameters.

Keywords: Service Level Agreement, Quality of Service, Service Level Specification, Semantic Web, Ontology, Computer Network, Network Service Provider.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SLA	*Service Level Agreement*
SLS	*Service Level Specification*
QoS	*Quality of Service*
NSP	*Network Service Provider*
CoS	*Class of Service*
WS	*Web Service*
SWS	*Semantic Web Service*
DiffServ	*Differentiated Services*
IntServ	*Integrated Services*
BE	*Best Effort*
IETF	*Internet Engineering Task Force*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
SLM	*Service Level Management*
RSVP	*Resource Reservation Protocolos*
DSCP	*Differentiated Services CodePoint*
PHB	*Per-Hop Behavior*
MF	*Multi Field*
AF	*Assured Forwarding*
EF	*Expedited Forwarding*
DF	*Default Forwarding*
BB	*Bandwidth Brokers*
VPN	*Virtual Private Network*
OWL	*Web Ontology Language*
RDF	*Resource Description Framework*
QoE	*Quality of Experience*
MOS	*Mean Opinion Score*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Com o aumento da competitividade no mercado em relação a aplicações de tempo real (VOIP, videoconferência, jogos, entre outros) e a crescente exigência dos consumidores para que os provedores desses serviços garantam a Qualidade de Serviço (QoS - *Quality of Service*) oferecida têm estimulado o desenvolvimento de mecanismos para a negociação e garantia da QoS ofertada. Para que tais consumidores tenham um Serviço com garantia de QoS, é necessário que estes especifiquem seus requisitos de qualidade.

A negociação desses requisitos e feita no momento de inscrição de um Serviço de rede e esta é acertada entre o Provedor do Serviço (NSP - *Network Service Provider*) e o cliente através de um contrato, denominado Acordo de Nível de Serviço (SLA - *Service Level Agreement*). Neste contrato além de serem definido os parâmetros mínimos de qualidade que o NSP deve garantir para o cliente, também são definidos o custo do fornecimento deste Serviço e as eventuais punições para o provedor caso a QoS acertada não seja respeitada ou para o cliente caso este faça mau uso do Serviço.

Uma SLA e composta por uma lista de Especificação de Nível de Serviço (SLS - *Service Level Specification*) que especificam um conjunto de parâmetros de qualidade, seus valores e suas métricas que variam de acordo com o tipo de Serviço prestado (GODERIS et al., 2002).

Atualmente não existe nenhuma normalização para a Especificação de QoS, assim como não existe uma lista de parâmetros de qualidade adotadas universalmente para compor uma SLS. Entre os exemplos de trabalhos onde houve uma tentativa de normalização na área de QoS estão os projetos TEQUILA (*Traffic Engineering for Quality of Service in the Internet, at Large Scale*)(GODERIS et al., 2002) e AQUILA (*Adaptative Resources Control for QoS Using an IP-based Layered Architecture*)(SALSANO et al., 2000) que, em concordância com outros trabalhos na área, adotam parâmetros da camada de rede referentes ao protocolo IP, como *One Way Delay*, *Packet Delay Variation* e *Packet Loss rate* e (GODERIS et al., 2002) que propôs uma lista de atributos para a definição da SLS.

A diversidade dos níveis de serviço, tecnologias e soluções de QoS fazem com que a adoção de uma lista pré-definida de parâmetros, para a definição da SLS, não seja o ideal. Isto exige que a especificação da QoS permita o uso de uma lista flexível de parâmetros e métricas (PRUDÊNCIO; WILLRICH, 2009). A princípio, no processo de negociação do serviço, o cliente e a aplicação devem ser capazes de especificar a QoS utilizando-se de parâmetros quantitativos e qualitativos. Parâmetros esses que devem ser compreendidos e mapeados de forma a serem reconhecidos pelos NSPs.

Trabalhos na área de serviços Web (WS - *Web Services*) enfatizam o uso de ontologias para a especificação semântica de serviços de uma maneira formal e passível de interpretação automatizada (BLEUL; WEISE, 2005; DOBSON; LOCK; SOMMERVILLE, 2005; ZHOU; CHIA; LEE, 2005). O uso de ontologias, definidas como um conjunto de conceitos, relações, instâncias e axiomas usados para descrever um domínio (DAVIES; STUDER; WARREN, 2006), fornece o vocabulário necessário para listar os aspectos funcionais e não funcionais de WS, tornando-se um mecanismo formal que permite ao usuário diferenciar o serviço baseado na sua funcionalidade e garantia da qualidade desejada. A linguagem padrão para criação de ontologia desenvolvida pela W3C e a Web Ontology Language (OWL) (OWL, 2004).

Diversas ontologias de QoS foram criadas na área de serviços Web Semânticos (SWS - *Semantic Web Services*) utilizando OWL para definir qualidade e diferenciar WS além do funcional (DOBSON e LOCK, 2005; ZHOU e CHIA e LEE, 2005; BLEUL e WEISE, 2005). Já na área de serviços de rede, Prudêncio (2010) propõe uma ontologia de QoS para serviços de rede, denominada NetQoSOnt, que será utilizada neste trabalho como base da abordagem semântica. Prudêncio et al (2009), afirma que a NetQoSOnt tem como meta principal “permitir a especificação da QoS com transparência em termos de parâmetros”(p. 2) e “oferece suporte a interpretação automática dos parâmetros de QoS através do uso de inferência em ontologias”(p. 2).

1.1 OBJETIVO GERAL

Durante o estabelecimento de um contrato SLA são especificados diversos parâmetros para o serviço, incluindo o nível de QoS que deve ser garantido. Em uma SLA, para um serviço de VOIP por exemplo, deve existir a possibilidade de escolha, por parte do cliente, da qualidade de voz desejada, quantidade de conexões simultâneas e o período que necessitará do serviço de voz. Para isso as especificações de qualidade devem ser mapeadas em parâmetros de desempenho de rede, a fim de que seja possível controlar a qualidade de serviço que a rede deverá prover para que seja mantida a qualidade de voz contratada pelo cliente (SILVA, 2005).

O objetivo do presente trabalho é desenvolver um protótipo para uma aplicação web para negociação de SLA, utilizando como base uma ontologia de SLA/SLS. Esta aplicação deverá permitir que, durante o período de validade da SLA, o usuário e a NSP possam criar, modificar e remover as SLSs, oferecendo assim um maior dinamismo na hora da definição da QoS e dando ao usuário um maior poder de escolha do nível de QoS desejado. Para testar o sistema web, este sistema será usando em uma arquitetura de gerenciamento de QoS para aplicações VoIP em uma rede DiffServ.

1.1.1 Objetivo Específicos

Para tanto foram traçados, para o presente trabalho, os seguintes objetivos específicos:

- Realizar um estudo sobre os conceitos relacionados a especificação de QoS em rede de computadores;

- Realizar um estudo sobre os conceitos relacionados a semântica computacional e ao desenvolvimento de ontologias.

- Implementar utilizando a linguagem Java uma aplicação web para a negociação de SLA;

- Utilizar a NetQoSOnt (PRUDENCIO, 2010), como ontologia base para especificação de QoS para serviços de rede;

- Utilizar a OWL API (HORRIDGE e BECHHOFER, 2009), uma API em Java, para a manipulação das ontologias

- Testar o sistema proposto em uma rede Linux Diffserv com um servidor VOIP Asterisk.

1.2 JUSTIFICATIVA

Os usuários e as novas aplicações em rede estão considerando essencial que os provedores de serviço possam garantir a QoS oferecida durante a negociação do serviço. Para isso a especificação da QoS pode ser feita utilizando-se parâmetros percebidos pelo usuário (como Bom, Ruim, entre outros) ou também parâmetros de rede ou da aplicação. No entanto, essa flexibilidade exige mecanismos eficientes de mapeamento de QoS. Além disso, a negociação de serviço utilizando-se de parâmetros dependentes da aplicação e complexa, pois se aplica a uma gama de serviços a serem negociados e cada serviço pode utilizar-se de diferentes parâmetros e métricas de QoS. Sendo assim soluções para a negociação de SLA deve oferecer flexibilidade em termos de parâmetros de QoS.

1.3 ESTRUTURA

Neste capítulo foi apresentada uma breve introdução do tema, bem como a motivação, escopo, objetivos e justificativa do presente trabalho. No capítulo 2 será

feita uma revisão dos conceitos gerais relacionados a especificação de QoS e também serão introduzidas as arquiteturas IntServ e DiffServ. Em seguida, no capítulo 3, será realizada uma revisão sobre os conceitos relacionados a serviços web semânticos, com contextualização da ontologia NetQoSOnt e apresentação da interface de programação OWL API. Continuando, no capítulo 4, será feito um estudo sobre Sistemas de Gerência de QoS e em seguida, no capítulo 5, será apresentado o protótipo proposto por este trabalho. Finalmente no capítulo 6, poderão ser encontradas as conclusões obtidas e também as perspectivas para trabalhos futuros.

2 QUALIDADE DE SERVIÇO

A medida que iniciaram o oferecimento de serviços de tempo real usando para tal serviços de comunicação de dados houve um grande aumento da necessidade de garantir a Qualidade de Serviço (QoS) ao nível dos serviços de comunicação de dados. Qualidade esta que está relacionada aos requisitos que uma aplicação necessita para garantir ao usuário um serviço que atenda as suas expectativas.

Diversas definições são propostas para QoS. Algumas delas podem ser encontradas em Marcheses (2007) e Park (2005). Prudêncio et al (2009) define QoS como “o grau com o qual um conjunto de características de um sistema atende a determinados requisitos”(página 22).

Os termos qualidade e serviço podem ser interpretados como a capacidade de um sistema de diferenciar e classificar diversos tipos de serviço, ou tipos de tráfego, de forma com que cada classe de serviço seja tratada de forma diferente. Uma rede TCP/IP básica é constituída de apenas uma classe de serviço onde o tráfego de dados é feito ponto a ponto sem que haja qualquer tipo de garantia de qualidade em termos de parâmetros como atraso(*One-way Delay),* taxa de perda de pacotes *(One-way Packet a Loss),* variação de atraso (*One-way IP Packet Delay Variation),* vazão (*Network Capacity)*, entre outros.

Esta classe de serviço é conhecida como Melhor Esforço (BE - *Best Effort*), onde todos os fluxos de dados são tratados da mesma maneira e assim, sem nenhuma garantia de tempo e de entrega de pacotes, os serviços de tempo real podem sofrer degradação na qualidade. Tendo isso em vista foram desenvolvidas novas arquiteturas para oferecer serviços previsíveis na Internet, com o objetivo de atender as necessidades de aplicações de áudio e vídeo, incorporadas a rede IP.

Segunda Royer (2008), as primeiras arquiteturas de gerenciamento de QoS propostas pela IETF *(Internet Engineering Task Force)* foram a IntServ que controla as reservas de recursos individualmente, por fluxo, e por isso tem problemas de escalabilidade, e a DiffServ, que apenas diferencia os tipos de aplicações em classes de serviço (CoS – *Class of Service*) com um tratamento diferenciado para cada classe

(e não individualmente por fluxo), nos roteadores, o que é escalável mas não oferece garantia plena de atendimento a cada fluxo. Estas duas arquiteturas originalmente não apresentavam um serviço de autorização de tráfego baseada em permissões de usuário. Ambas as arquiteturas serão estudadas com mais detalhes mais adiante neste mesmo capítulo.

Outra característica importante de serviços de rede e que deve ser levada em consideração para o tratamento de QoS é a diferença entre a largura de banda disponível no centro e na borda da rede, que faz com que os roteadores de borda tenham tipicamente mais congestionamento do que os localizados no centro, gerando assim uma grande necessidade de realizar classificação dos pacotes, oferecendo proteção e priorização para determinados fluxos de dados, nos roteadores de borda.

2.1 PARÂMETROS DE QUALIDADE DE SERVIÇO

Para a especificação da qualidade do serviço de uma determinada aplicação, é necessária a definição de um conjunto de parâmetros de qualidade de serviço que deverá ser garantido ao fluxo de pacotes.

Os parâmetros mais importantes para controle do sistema são os seguintes:

- **Vazão *(Network Capacity):*** é a taxa efetiva de bits trafegada na rede, medida em bits/s. Pode-se dizer que é a capacidade do sistema em transferir dados. A vazão da maioria das redes varia com o tempo, já que os recursos da rede são utilizados de forma estatística. Também podem ocorrer congestionamentos em determinadas rotas, ocasionando menor vazão durante esses períodos.

- **Atraso *(One-way Delay):*** é o tempo total entre o envio de um pacote, ou mensagem, a partir da origem até sua recepção no destino. É a somatória dos atrasos gerados pelos meios de transmissão e pelos equipamentos envolvidos na rota ponto-a ponto da transmissão,

incluindo roteadores, switches, modems, etc. Do ponto de vista da aplicação, a latência (atraso) resulta em um tempo de resposta ou tempo de entrega da informação. Os principais fatores que influenciam na latência são: o atraso de propagação do sinal no meio de transmissão; a velocidade da transmissão; e o tempo de processamento nos equipamentos da rede. Em redes WAN o efeito da latência é muito mais observado que em redes LAN. Os próprios hosts e servidores também geram atrasos que devem ser considerados se a medição for ponto-a ponto na aplicação.

- **Variação de Atraso *(One-way IP Packet Delay Variation):*** pode ser visto como a variação do atraso de entrega dos pacotes de um mesmo fluxo. A variação de atraso é gerada principalmente pela variação nos tempos de processamento dos pacotes pelos equipamentos de rede, como roteadores. Isto ocorre, em geral, devido às variações na quantidade de tráfego que cursa pela rede. Aplicações interativas de voz e vídeo, ou aplicações de tempo-real podem ser afetadas pela variação do atraso.

- **Taxa de perda de pacotes *(One-way Packet a Loss):*** é a razão entre a quantidade de pacotes perdidos na rede e a quantidade de pacotes que foram transmitidos. Esta perda ocorre principalmente devido ao descarte de pacotes nos roteadores, durante períodos de congestionamentos, onde os buffers internos não comportam a quantidade de pacotes a armazenar. O processo de descarte depende de fatores como condicionamento do tráfego, enfileiramento e tratamento dos pacotes nos buffers internos. Também podem ocorrer perdas de pacotes no meio de transmissão, porém a grande incidência de taxa de erros está na última milha, onde normalmente é utilizado par metálico, sendo mais sensível a falhas.

2.2 ESPECIFICAÇÃO DE QOS

Uma especificação de QoS é constituída por um conjunto de parâmetros que, em conjunto com seus valores, definem a qualidade de serviço oferecida por uma aplicação ou serviço (GROSSMAN, 2006). E pode ser utilizada tanto pelo provedor, para descrever a qualidade ofertada, ou pelo cliente, para descrever a qualidade desejada.

A especificação de QoS apresenta grande importância em diversas momentos no gerenciamento de qualidade em redes. Como, por exemplo, no momento de inscrição de um serviço, na negociação da qualidade, na invocação do serviço e também no monitoramento do serviço (PRUDÊNCIO, 2010).

No momento de inscrição de um serviço é onde ocorre a negociação de um contrato de serviço, denominado Acordo de Nível de Serviço ou SLA (do inglês, *Service Level Agreement)*. Em uma SLA encontram-se aspectos não técnicos, como a identificação das partes, custo do serviço, punições para ambas as partes em caso de descumprimento do contrato, entre outros. E também aspectos técnicos (parâmetros de qualidade em conjunto com seus valores e métricas), denominados Especificação de Nível de Serviço ou SLS (do inglês, *Service Level Specification)*, que devem ser garantidos pelo provedor do serviço para satisfação do cliente.

Esta negociação geralmente é feita manualmente e tem como resultado os chamados SLAs estáticos. Mas essa negociação também pode ocorrer de maneira dinâmica. Sendo necessário que ambas as partes estabeleçam uma sintaxe para a negociação automática do SLA, gerando os chamados SLAs Dinâmicos. Neste caso, diferentemente dos SLAs estáticos, existe a possibilidade de frequentemente modificar especificações do contrato (mudanças previamente definidas neste mesmo contrato), de acordo com a necessidade atual do cliente e da disponibilidade do provedor (MESCAL apud PRUDÊNCIO, 2010).

Antes da utilização de um serviço ocorre a Invocação do Serviço e pode ser feita de maneira implícita e explícita (TEQUILA apud PRUDÊNCIO, 2010). No caso de uma invocação implícita defini-se a QoS a ser garantida em uma SLA negociada

previamente e, em alguns casos, são necessários protocolos de sinalizacão de QoS para reserva de recursos. Já no caso de uma invocação explícita o usuário ou aplicação especificam o nível de QoS e o provedor pode ou não aceitar o pedido, com base na carga do sistema.

No Monitoramento do Serviço assegura-se a detecção de violação no contrato, tanto por parte do cliente quanto por parte do provedor. Sendo ideal que uma terceira entidade (uma empresa de segurança, por exemplo) seja responsável por monitorar a utilização dos recursos pelo cliente e também o provimento destes pelo provedor, alertando ambas as partes caso seja encontrada uma violação (MORAES et al., 2008). E em caso de uma violação ser encontrada, esta deve ser tratada conforme descrito na SLA previamente estabelecida.

2.3 ACORDOS DE NÍVEL DE SERVIÇO

Verma99 apud Pras, Sprenkels apresenta uma definição geral para Acordos de Nível de Serviço (*Service Level Agreement - SLA):*

"Uma declaração explícita de expectativas e obrigações existentes em uma relação de negócios entre duas organizações: o prestador de serviços(Network Service Provider - NSP) e o cliente."

Uma SLA representa um contrato formal sobre o serviço a ser entregue e é utilizada por ambas as parte envolvidas; o provedor do serviço a utiliza para ter um controle do serviço que deve ser oferecido e, em caso de disputas legais com o cliente, esta pode ser utilizada como prova em caso de reclamações ou mau uso do serviço por parte do cliente. O mesmo ocorre no sentido contrário; o cliente também usa a SLA como um descritor formal sobre o serviço que o provedor deve entregar e pode utilizá-la caso exista uma quebra de contrato por parte do provedor.

Normalmente uma SLA apresenta os seguintes componentes (Verma99 apud Pras , Sprenkels):

- descrição do serviço a ser oferecido;
- a qualidade esperada do serviço;
- os procedimentos caso existam problemas com o serviço;
- um procedimento para monitorar e reportar ao cliente a QoS esta esta sendo utilizada;
- as consequências para o provedor do serviço em caso de não cumprimento da QoS acordada. O Mesmo para o cliente em caso de mal uso do serviço;
- uma descrição sob quais circunstâncias a SLA não se aplica.

O cliente utiliza a SLA para verificar se esta recebendo o serviço no nível acordado e pode reportar ao provedor de serviço caso perceba que a qualidade de serviço não está de acordo com o contrato. Isto é possível desde que a SLA contenha parâmetros pré-determinados que devam ser expostos para o cliente , a fim de que este monitore a QoS em tempo real, e também um mecanismo para que o cliente possa reportar um problema ao provedor de serviço.

Diante da diversidade dos serviços de rede configuráveis existe um grande conjunto de parâmetros do serviço que podem ser monitorados. Valores para esses parâmetros devem ser determinados baseados nas preferências do cliente. Em um sistema de gerenciamento de SLA, proposta deste trabalho, o cliente assume controle deste processo e pode definir valores para alguns parâmetros de forma dinâmica dentro do limites pré-determinados em uma SLA.

2.3.1 Ciclo de Vida de uma SLA

Pras, Sprenkels definem três fases durante a existência de uma SLA: fase de criação, fase operacional e fase de remoção.

- **Fase de Criação:** nesta fase o cliente assina um contrato de entrega de serviço oferecido por uma organização. Neste momento o cliente esta

ciente do que foi legalmente acertada a respeito da qualidade do serviço que está sendo contratado.

- **Fase Operacional:** nesta fase o cliente tem acesso aos termos estáticos de uma SLA (e.g. Eventuais multas devido a mau uso do serviço) e também pode realizar mudanças nos parâmetros de especificação de nível de serviço (*Service Level Specification – SLS)* de acordo com o que foi pré-determinado na SLA. Para isso, o cliente necessita de uma interface onde possa alterar as SLS que fazem parte desta SLA. Qualquer alteração que o cliente faça deve ser mapeada para a configuração atual do serviço e isto implica, na maioria das vezes, em quanto o cliente pagará por esse serviço. A interface mencionada e o mapeamento da nova configuração são assuntos abordados pelo protótipo apresentado no capítulo 5.

- **Fase de Remoção:** fase final do ciclo de vida da SLA onde, ao terminar o contrato, toda configuração associada aquele serviço deve ser removida.

2.3.2 Especificação De Nível De Serviço

Os requisitos de qualidade estão definidos na Especificação do Nível de Serviço (SLS – Service Level Specification), onde são definidos os valores dos parâmetros que o provedor se compromete em fornecer ao seu cliente. O SLS especifica os parâmetros de qualidade de serviços que serão fornecidos aos clientes, isto é, a especificação da classe de serviço e suas características.

Goderis et al. (2002) define alguns elementos para compor uma SLS:

- Vazão (banda): taxa de transmissão de pacotes garantida;

- Atraso da rede: tempo de atraso entre 2 pontos da rede;

- Jitter: variação do atraso entre a chegada de pacotes;

- Perda de pacotes: é a taxa de perda de pacotes em relação a quantidade de pacotes transmitidos;

- Disponibilidade: é o percentual de tempo em que o serviço está ativo, e dentro dos parâmetros contratados pelo cliente, que normalmente são medidos mensalmente ou anualmente;

- Tempo máximo de solução de problemas: é o tempo em que o provedor leva para recuperar uma falha no serviço prestado;

- Períodos de garantia dos parâmetros de QoS: são os períodos do dia ou mês, onde o provedor se compromete em fornecer a qualidade contratada, por exemplo nos dias úteis das 08:00 as 18:00hs;

- Local onde é disponibilizado o serviço: é o endereço físico, tipo localidade, rua, avenida, numero, onde o provedor entrega o serviço;

- Quais são as informações utilizadas para marcar os pacotes, por exemplo portas UDP ou TCP, porta física de acesso ao provedor, endereço de origem ou destino;

- Como são classificados os fluxos de pacotes de acordo com as marcações previamente definidas;

- Como são definidos os perfis de cada fluxo de pacotes, por exemplo: taxa de pico, taxa média e tempo máximo do tráfego no pico;

- Qual é o tratamento aos pacotes que estão fora do perfil dos fluxos, que poderão ser descartados, conformados/suavizados (atrasados até que se situem dentro do perfil desejado), ou remarcados para outro tipo de fluxo.

Para avaliar e validar se o provedor esta cumprindo o contrato SLA, este deve comprovar junto ao cliente as medições destes níveis. Para disponibilizar estas informações, o provedor deverá possuir ferramentas e metodologias necessárias para medir o desempenho do serviço prestado. Estas ferramentas e metodologias são chamadas de Gerência de Nível de Serviço (SLM – S*ervice Level Management*).

A Gerência de Nível de Serviço deve monitorar e controlar todos os processos que se relacionam com a qualidade negociada no contrato.

2.4 ARQUITETURA DE QOS

Em redes de computadores deve-se levar em consideração um mecanismos ponto a ponto de garantia de entrega de informações na implementação de uma QoS. Desta forma, implementar QoS em rede significa atuar em diversos equipamento envolvidos na comunicação ponto a ponto visando o controle dos parâmetros definidos (SILVA, 2005) .

O IETF *(Internet Engineering Task Force)* tem proposto diversos modelos e mecanismos para satisfazer a necessidade de QoS na Internet. Possibilitando um melhor controle sobre o tráfego priorizando certas aplicações em detrimento do restante. Entre estes trabalhos encontram-se os modelos MPLS (MultiProtocol Label Switching) (ROSEN, VISWANATHAN e FELDMAN, 2001), Serviços Integrados/RSVP (IntServ) (BRADEN, CLARK e SHENKER, 1994) e Serviços Diferenciados (DiffServ) (BLAKE et al., 1998).

2.4.1 MPLS

No caso da solução MPLS existe uma certa relação com a questão da qualidade em serviços de redes, pois um dos ganhos mais importantes com sua utilização é a simplificação da função de roteamento, reduzindo o overhead e as latências nos roteadores. É uma solução mais orientada para uma engenharia de tráfego de pacotes na rede do que para qualidade de serviço.

Esta solução realiza a marcação dos pacotes com um rótulo (*label* MPLS) nos roteadores de borda de uma rede que serão verificados pelos roteadores internos a fim de verificar para onde o pacote deve ser encaminhando. Com estes rótulos é

possível estabelecer um caminho, de forma a utilizar as melhores rotas obtendo um melhor desempenho para o fluxo de dados.

Com a utilização do MPLS consegue-se uma melhoria na qualidade do serviço devido a melhores condições de operação da rede, mas este não provê controles específicos quanto a garantia de QoS em uma rede de computadores (SILVA, 2005).

2.4.2 Serviços Integrados – IntServ

No modelo IntServ (Integrated Services) Braden (1994), estabelece a implantação de uma infraestrutura para a Internet com o intuito de suportar o transporte de áudio, vídeo e dados em tempo real além do tráfego de dados atual. Nesta arquitetura a QoS e garantida através de mecanismos de reserva de recurso na rede, onde a aplicação deve reservar os recursos que deseja utilizar antes de iniciar o envio dos dados, usando o protocolo RSVP (Resourse Reservation Protocol).

O protocolo de sinalização RSVP atua sobre o tráfego de pacotes em uma rede TCP/IP (Internet, redes privadas, entre outras). É considerado um protocolo eficiente em relação a garantia de QoS a medida que provê granularidade e controle fino das solicitações feitas pelas aplicações, tendo como maior desvantagem a sua complexidade de operação nos roteadores que, eventualmente, podem gerar problemas nos backbones de redes maiores, como das Operadoras (SILVA, 2005).

Uma questão a ser observada para definir mecanismos de qualidade de serviço, é a de que a necessidade de garantir QoS se mostra mais importante nos períodos de pico de tráfego, quando a rede enfrenta uma situação de congestionamento ou de carga muito elevada. Nesta situação os mecanismos de QoS devem buscar soluções para decisões do tipo:

- Como alocar os escassos recursos, como banda de transmissão;
- Como selecionar o tráfego de pacotes;

- Como priorizar os pacotes;

- Como descartar pacotes, ou seja, quais e quando descartá-los.

2.4.3 Serviço Diferenciados – DiffServ

A arquitetura DiffServ *(Differentiated Services)* (BLAKE, 1998) foi introduzida pelo IETF devido as limitações na arquitetura IntServ. No DiffServ os pacotes são marcados diferentemente, na entrada da rede, para criar classes de pacotes. As classes recebem serviços, ou tratamentos diferenciados na rede, conforme definições preestabelecidas. Serviços Diferenciados é essencialmente um esquema de priorização de pacotes.

Um serviço pode ser caracterizado por alguns aspectos significativos relacionados à transmissão dos pacotes, de acordo com parâmetros como vazão, atraso, variação de atraso ou perda de pacotes. Além da arquitetura DiffServ atender aos requisitos de QoS de aplicações heterogêneas, e diferentes expectativas de usuários, também permite uma tarifação diferenciada para os serviços providos pelas Operadoras.

Nesta arquitetura, a diferenciação dos serviços de rede é feita através da especificação e marcação do campo DS do cabeçalho do data grama IP. Os Serviços Diferenciados são oferecidos no interior de um domínio DS, que é composto por um conjunto de nós compatíveis com a proposta DiffServ. No DiffServ, a marcação do campo DS indica o tipo de serviço que deverá ser provido pelos sistema de comunicação para o pacote.

Os serviços oferecidos por um domínio DS são todos para tráfego unidirecional e para tráfegos agregados, não fluxos individuais. Os tráfegos agregados também são chamados de comportamentos agregados (BA - *Behavior Aggregate*), e são definidos como conjuntos de pacotes que tem a mesma marcação no campo DS, e

que atravessam um caminho na mesma direção. Esses pacotes podem vir de origens ou aplicações distintas.

Domínios DS também podem ser concatenados, ou seja, domínios diferentes podem estar ligados entre si. Também podem estar conectados a domínios não DS, tomando como exemplo uma conexão de uma rede DS de uma Operadora com uma rede não DS de um Cliente.

Os serviços entre os domínios podem ser definidos em uma SLA, que especifica a forma de encaminhamento e garantias de prestação do serviço através do domínio DS. Na figura 1, é apresentada a arquitetura geral de domínios DS, onde podemos ver nós de borda (ingresso e egresso), nós internos (interiores), domínios DS, e onde é aplicado o SLA e SLS (SILVA, 2005).

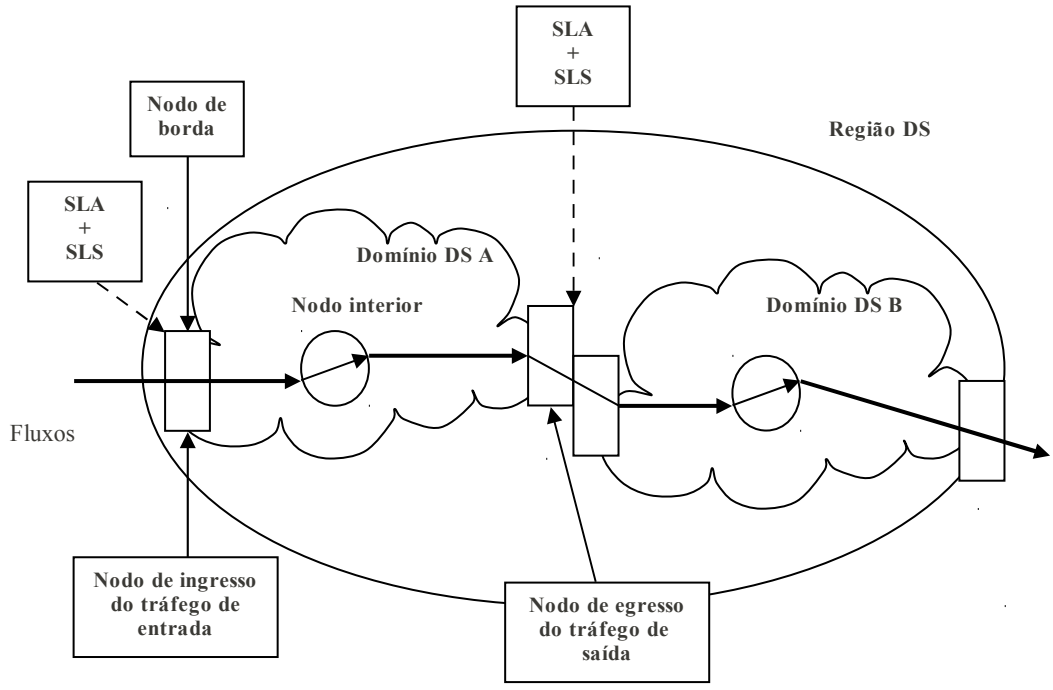

Figura 1 – Arquitetura geral de Domínios DS

No ingresso ao domínio DS, o roteador de borda classifica os pacotes e realiza o condicionamento de tráfego para torná-los conformes ao perfil de tráfego previamente contratado. As regras de classificação e condicionamento, assim como montante de espaço de bufferização e largura de banda nos roteadores são

derivadas do SLA, mais especificamente do SLS, de onde se constrói todo o perfil de tráfego do cliente. Um exemplo simples de perfil de tráfego poderia ser: medir o fluxo de pacotes do endereço IP a.b.c.d., e se sua taxa ficar abaixo de 384 Kbps, setar o campo DS para o valor X, senão setar o campo DS para o valor Y. Se a taxa exceder 512 Kbps, descartar os pacotes excedentes.

O campo DS é formado por 8 bits (veja figura 2), sendo que os 6 primeiros bits compõem o DSCP (Differentiated Services CodePoint), definido na RFC-2474 (NICHOLS, 1998), que é utilizado para definir o comportamento por salto, ou PHB (Per-Hop Behavior), que o pacote sofrerá em cada nó. Este campo é definido como um campo não estruturado para facilitar a definição de futuros PHBs, suportando até 64 valores diferentes, ou codepoints. O campo DS também possui dois bits CU (Currently Unused) reservados para o futuro.

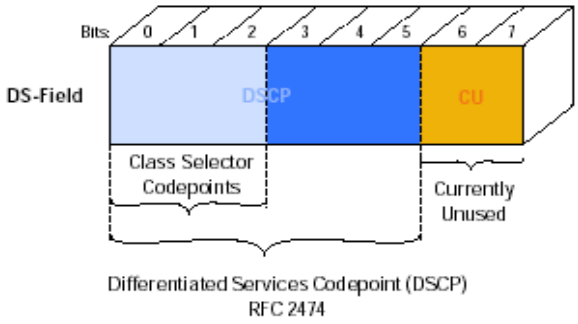

Figura 2 - Estrutura do Campo DS

A marcação deste campo pode ser realizada no roteador que liga o cliente a rede do provedor do serviço (roteador de borda), baseada na classificação MF (Multi Field) ou também pode ser realizada diretamente pela aplicação do cliente.

Desta forma, é possível classificar os serviços de rede em Classes de Serviço ou CoS (*Class of Services).* O conjunto de CoS definidos por DiffServ, identificados pelo campo DSCP do pacote IP, é apresentado a seguir (BARBIARZ, CHAN e BAKER, 2006):

• **Default Forwarding (DF):** é a CoS padrão, e especifica que os pacotes marcados com esse DSCP receberão o tratamento de Melhor Esforço ou Best Efort, garantindo uma vazão mínima de tráfego, mas sem garantia de ordem ou entrega dos pacotes;

• **Assured Forwarding (AF):** sugerida para aplicações que necessitam de confiabilidade e prioridade maior no tráfego de dados com relação ao que é oferecida pelo serviço de Melhor Esforço. É na verdade um conjunto de CoSs, com diferentes níveis de garantia em atraso, variação de atraso, perda e vazão. Cada classe deverá exibir um comportamento diferente quanto ao método de encaminhamento, independente do comportamento de outras classes, sendo que cada uma deverá apresentar níveis de precedência quanto a decisão de descarte de pacotes.

• **Expedited Forwarding (EF):** é a CoS de maior prioridade e foi designada para oferecer serviços ponto a ponto com baixo atraso, baixa variação de atraso, baixa perda de pacotes e vazão garantida.

Em uma arquitetura DiffServ com *Bandwidth Brokers (BB)* (NICHOLS apud ROYER, 2008), quando uma máquina precisa submeter um tráfego com QoS para a rede, ela solicita ao BB (enviando uma solicitação ao roteador de borda) qual a classe de serviço (CoS) que deve marcar em seus pacotes. Quando um roteador de borda recebe uma requisição de QoS, ela é encaminhada para o BB, que verifica: (i) as credenciais do usuário (autenticação), (ii) se o fluxo total para esse usuário, incluindo a nova requisição não é maior do que a largura de banda contratada (autorização) e (iii) se a rede tem recursos suficientes para suportar a nova requisição (CAC). Se todas as três condições forem satisfeitas, uma mensagem é enviada para os roteadores envolvidos para reservar a largura de banda para o fluxo, e a quantidade de largura de banda requisitada é decrementada da quantidade disponível, e registrada como usada pelo usuário.

A arquitetura BB suporta dois tipos de serviço, além do melhor esforço: serviço *premium*, e serviço garantido (*assured*). O serviço premium recebe um tratamento de encaminhamento expresso, prioritário em relação a todos os outros, independente do tamanho das filas dos demais serviços. A reserva de banda para esse tipo de serviço

é efetuada pela taxa de pico, e não são admitidas rajadas, dispensando ou minimizando a ocorrência de filas nos roteadores. Caso a taxa de tráfego submetida seja maior do que a taxa prevista, os pacotes excedentes são descartados. Dessa forma, a variação de atraso para esse tipo de pacote é minimizada, tornando-o adequado a serviços como VoIP, e VPN. Em contrapartida, por ter recursos alocados pela taxa de pico, muitas vezes está subutilizado, e usualmente é mais caro. Já os serviços garantidos possuem diferentes níveis de prioridade, e apresentam a mesma característica de variação do atraso do serviço de melhor esforço, com a diferença de que seus pacotes têm mais prioridade. A intensidade dessa variação e a garantia de desempenho dependem do ajuste do provisionamento de banda para cada classe, dos tamanhos das rajadas aceitas e dos tamanhos das filas alocadas. Caso a taxa de submissão de uma rajada de tráfego garantido seja maior do que a taxa prevista, o excedente é remarcado como melhor esforço. Em casos eventuais de redes congestionadas, alguns pacotes podem ser descartados, o que pode ser tolerado por muitos usuários.

Estes serviços quantitativos oferecem garantias concretas que podem ser avaliadas por medições convenientes, independentes de outros serviços, por exemplo:

- 90 % do tráfego dentro do perfil para o serviço “C”, terá menos que 50 ms de atraso;

- 95 % do tráfego dentro do perfil para o serviço “D”, será efetivamente entregue no destino (perda de pacotes igual a zero).

Também pode ser feita uma quantificação relativa, por exemplo: o tráfego oferecido no nível de serviço “E” terá três vezes a largura de banda que o tráfego oferecido no nível de serviço “F”.

A definição dos serviços, ou classes de serviços, que serão prestados pelo provedor ficam a seu critério, mas de uma forma geral pode-se dizer que um portfólio completo inclui serviços para voz, vídeo e dados com alguns níveis de prioridade. Segundo (Retiere, 2002), além do melhor esforço poderiam ser disponibilizadas cinco classes de serviço como:

- Classe Voz – serviço otimizado para aplicações em tempo real, muito sensíveis ao tempo, como VoIP e sistemas de controle. Classe implementada com PHB EF.

- Classe Vídeo Interativo – serviço para aplicações em tempo real de vídeo, como videoconferência. Classe implementada com PHB AF41.

- Classe Interativa – serviço para aplicações de negócios que exijam rápido tempo de resposta, consideradas críticas. Classe implementada com PHB AF31.

- Classe Negócios – serviço adequado para aplicações transacionais, comuns de negócios, como tráfego cliente/servidor e web corporativa. Classe implementada com PHB AF21.

- Classe Geral – serviço de mais baixo custo, para tráfego de negócios não críticos como e-mail e streaming de vídeo, como vídeo sob demanda ou e-learning por exemplo. Classe implementada com PHB AF11.

Nesta arquitetura especifica-se quais as classes de serviço e quais mecanismos que podem ser utilizados para classificar e moldar o tráfego de acordo com as classes mencionadas anteriormente (DF, AF e EF). Mas cada provedor pode especificar uma configuração real para estes mecanismos, abrindo margem para interpretações diferentes dos padrões (como por exemplo uma classe EF para um provedor pode oferecer a mesma garantia que a classe AF1 de outro provedor) (PRUDÊNCIO, 2010).

3 SEMÂNTICA COMPUTACIONAL

Com a popularização da Internet e da World Wide Web os usuário passaram a ter acesso a uma grande quantidade de dados, gerando uma “sobrecarga de informação”, dificultando o gerenciamento e a habilidade de tirar conclusões, de forma manual, diante de tanta informação. Além disso, diante do surgimento de novas tecnologias de desenvolvimento de sistemas computacionais, fez-se necessário lidar com informação legada e, com a terciarização de serviços, surgiu a necessidade da interoperabilidade de dados entre provedores (DAVIES, STUDER e WARREN apud PRUDÊNCIO, 2010).

Desta forma passou a existir a necessidade de automatizar o gerenciamento e a filtragem de dados disponíveis na Web, com o intuito de utilizá-los de forma eficiente. Para um gerenciamento eficiente desta gama de informações é preciso descrevê-la de maneira a torná-la compreensível por sistemas computacionais. E este é o papel da semântica, que permite uma descrição formal da informação de modo que esta seja gerenciada de forma automática ou semiautomática por computadores.

Com a descrição formal das informações é possível estabelecer equivalências entre conceitos sintaticamente distintos, permitindo a criação de mapeamentos entre terminologias de provedores diferentes (DAVIES, STUDER e WARREN apud PRUDENCIO, 2010). Estes conceitos são descritos de maneira semântica utilizando-se de ontologias, que, inspiradas no conceito homônimo da Filosofia e baseadas em formas de Lógica Computacional, permitem embutir significado em informações através de metadados que são compreensíveis por sistemas computacionais.

O objetivo deste capítulo e apresentar de forma geral os conceitos relacionados a especificação de QoS através de ontologias tanto para Serviços Web quanto para Serviços de Comunicação. Assim como apresentar a ontologia de especificação de QoS, denominada NetQoSOnt, utilizada por esse trabalho.

3.1 ONTOLOGIAS

Na área da computação, Ontologias podem ser definidas como um conjunto de conceitos, relações, instâncias e axiomas usados para descrever um domínio de interesse (DAVIES, STUDER e WARREN apud PRUDÊNCIO, 2010). Um conceito, também chamado de classe, pode ser definido como uma representação da informação modelada e pode conter uma ou mais instâncias (e.g. o conceito Cidade pode ter uma instância chamada Florianópolis), também chamada de indivíduos. Um Indivíduo pode relacionar-se com outro indivíduo ou com tipos primitivos (como string ou números), relação esta denominada propriedade. E axiomas são restrições impostas aos conceitos e as relações.

3.1.1 *Web Ontology Language – OWL*

A *Web Ontology Language (OWL)* é a linguagem padrão da W3C para a criação de ontologias (W3C, 2004a). Derivada do *Resource Description Framework (RDF),* e serializada em XML, OWL provê construções para criar tudo o que foi citado anteriormente: classes, instâncias, relações entre classes ou instâncias, chamadas de propriedades de objeto (*object properties),* relações entre classes/instâncias e tipos primitivos, chamadas propriedades de tipos de dados (*datatype properties)* e axiomas – utilizados para definir o domínio *(domain)* e o alcance *(range)* das propriedades, por exemplo. Todas estas construções podem ser denominadas genericamente de recursos (do inglês *resources*), terminologia também utilizada em RDF.

OWL possui três sublinguagens. OWL Lite é a mais restrita, recomendada para usuários que precisam apenas de uma hierarquia de classificação e restrições simples, e é computacionalmente decidível. OWL DL é um superconjunto de OWL Lite, baseada em um tipo de lógica de primeira ordem chamado Lógica Descritiva. OWL DL permite maior expressividade sem comprometer a computabilidade. Finalmente, OWL Full é um superconjunto de OWL DL, que permite máxima expressividade, porém sacrifica a decidibilidade da ontologia descrita (W3C, 2004).

Existem duas versões de OWL sendo utilizadas. A OWL 1.0 é o padrão atual, recomendação oficial da W3C. E a OWL 2.0 que é uma atualização de OWL 1.0, atualmente no *status* de Recomendação Candidata. OWL 2.0 traz novas funcionalidades, e já é implementada em várias ferramentas que trabalham com ontologias, tais como os motores de inferências FACT++ (TSARKOV; HORRCKS, 2009), Pellet (C&P, 2009), API como a OWL API(MANCHESTER U., 2009) utilizada por este trabalho e descrita mais adiante, e editores gráficos como o Protégé (BMIR,2009).

Em OWL, ontologias estão sujeitas à chamada pressuposição de mundo aberto (*Open World Assumption).* Esta pressuposição define que se um recurso não pode ser declarado verdadeiro utilizando o conhecimento existente na ontologia, este recurso não pode ser declarado falso. Uma consequência disso é que conceitos ou indivíduos diferentes devem ser explicitamente declarados com tal. Por esta razão, OWL também provê axiomas para declarar conceitos como disjuntos, e indivíduos ou propriedades como diferentes.

Outra funcionalidade importante de OWL que diz respeito à diferença entre recursos é que não existe suposição de nomes únicos (do inglês *Unique Name Assumption,* referido também pela sigla UNA): dois ou mais recursos diferentes podem ter o mesmo nome através de ontologias diferentes, e isso não significa que são a mesma coisa. Na verdade, o que identifica um recurso em uma ontologia é seu URI, composta pelo URI base da ontologia mais o URI do recurso. Esse sistema de identificação foi herdado de RDF, assim como vários outros recursos de metalinguagem (rótulos, comentários, entre outros).

3.1.2 OWL API

Esta secção apresenta a OWL API (HORRIDGE e BECHHOFER, 2009). Uma interface de programação de aplicações (do inglês Application Programming Interface – API), escrita em JAVA, para acessar e manipular ontologias OWL. A ultima versão desta API tem como foco a OWL 2 e esta suporta a análise e processamento de

ontologias nas sintaxes definidas pela W3C. Ou seja, *Functional Syntax*, RDF/XML, OWL/XML e *Manchester OWL Syntax.*

A ultima versão da OWL API foi projetada para atender as necessidades de pessoas que desenvolvem aplicações baseadas em OWL, editores OWL e *reasoners* OWL. Ela inclui suporte para mudanças em classes, interfaces gerais para trabalhar com *reasoners,* validadores para os diversos perfis da OWL 2, e suporte para análise e serialização de ontologias em diferente sintaxes.

Sua primeira versão, apresentada em (BECHHOFER, VOLZ e LORD, 2003), oferece um conjunto de interfaces juntamente com referências de implementações que facilitam o uso da OWL em diversas aplicações. Embora sua ultima versão apresente grandes semelhanças com a primeira, diversas mudança foram realizadas para alinhá-la a especificações da OWL 2 MOTIK, PATEL-SCHNEIDER, PARSIA apud HORRIDGE e BECHHOFER (2009).

Gerenciamento de Ontologias

Em sua nova versão além do modelo de interfaces para representar entidades, expressões de classe e axiomas, foi inserido um mecanismo que permite que as aplicações possam gerenciar as ontologias. A figura 3 mostra uma visão geral deste cenário.

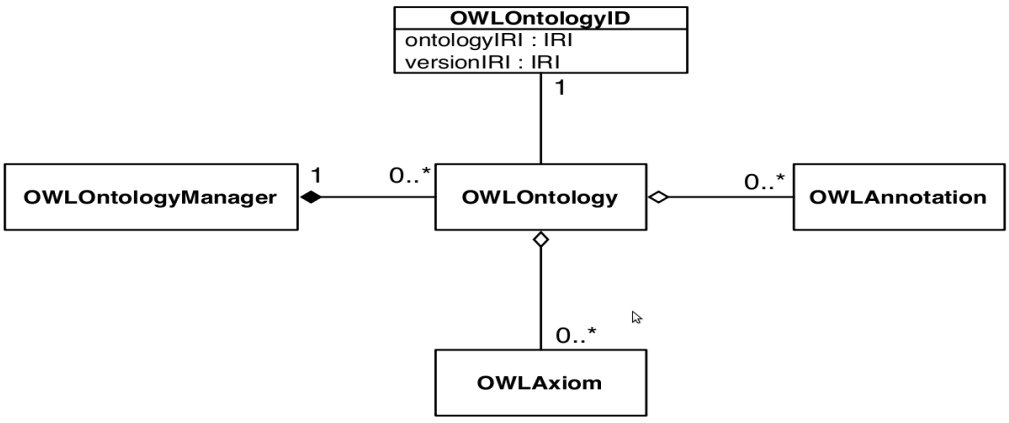

Figura 3 - Diagrama UML mostrando o gerenciamento de ontologias com a OWL API

A interface *OWLOntology* oferece um mecanismo para acessar os axiomas contidos em uma ontologia. Por exemplo, axiomas podem ser acessados por tipo ou

por assinatura do nome. A interface *OWLOntologyManager* provê mecanismos para criar, carregar, mudar e salvar as ontologias, que sejam uma instância de *OWLOntology.* Cada ontologia é criada ou carregada por um *ontology manager*. Cada instância de uma ontologia é única para um *manager* em particular, e todas as mudanças em uma ontologia são realizadas pelo seu *manager.* Desta forma as aplicações possuem um ponto de acesso às ontologias, e estas podem monitorar as mudanças realizadas em qualquer ontologia carregada.

Mudanças nas ontologias

Mudanças em ontologias podem ser realizadas por objetos que implementem a interface *OWLOntologyChange.* Além de adicionar e remover axiomas, mudanças em ontologias incluem atribuições de um *ID* para uma ontologia, adição e remoção de anotações, e adição e remoção de *imports.*

Todas as mudanças são aplicadas através do *ontology manager.* Isto significa que é possível aplicar uma lista de alterações, que façam múltiplas mudanças em múltiplas ontologias, como se fosse uma única unidade. Por exemplo no caso da alteração do nome de um entidade, que pode envolver adições e remoções de múltiplos axiomas em múltiplas ontologias, é possível agrupar as mudanças formando uma “operação de edição” e aplicar essas mudanças de uma vez só.

Interface *Reasoner*

Um *reasoner* semântico ou simplesmente *reasoner* é utilizado para checar a consistência de uma ontologia. Ou seja, um *reasoner* é capaz de, logicamente, inferir consequências diante de um conjunto de fatos ou axiomas.

A OWL API contém várias interfaces para suportar a interação com *reasoners.* A principal interface é *OWL Reasoner,* que provê métodos para executar as tarefas antes mencionadas. Estas interface foram projetadas para que os *reasoners* possam expor funcionalidades que proveem suporte para o raciocínio incremental.

Exemplos de uso

Alguns exemplos da utilização da OWL API (HORRIDGE e BECHHOFER, 2009):

- Protégé : um editor software livre de ontologias OWL que foi inicialmente projetado e desenvolvido pela Universidade de Manchester. Protégé-4 utiliza a OWL API para sustentar todas as tarefas de gerenciamento de ontologias, carregar e salvar ontologias, manipular durante a edição, e também oferecer opção da escolha por algum tipo de *reasoner;*

- **NeOn Toolkit :** um ambiente de desenvolvimento de ontologias baseado no Eclipse;

- OWLSight : web browser para ontologias escrito por Clark & Persia que utiliza o reasoner Pellet. O *browser* é escrito utilizando o *Google Web Toolkit,* com a OWL API sendo utilizada para acessar e ler ontologias;

- **OntoTrack :** uma ferramenta de navegação e edição de ontologias OWL que é desenvolvida na Universidade Ulm. A OWL API é utilizada para carregar e acessar ontologias para leitura em um gráfico.

3.2 ONTOLOGIA DE QOS PARA SERVIÇOS WEB

Uma abordagem semântica para a negociação de QoS não e assunto novo na área de Serviços Web (WS - Web Services). Os requisitos de qualidade para um WS, tratado dentro do contexto de um SLA, pode ser especificada utilizando uma linguagem especifica (e.g. WSOL - Web Service Offerings Language) para prover suporte sintático para a especificação.

E para comparar e equivaler esta especificação e necessário a expertise de um desenvolvedor. Deixando claro que uma abordagem semântica, utilizando-se de ontologias, poderia facilitar o uso de WSs.

Uma das principais vantagens de se utilizar ontologias é de que nem todo conhecimento precisa ser explicitamente descrito e novas relações entre conceitos e instâncias podem ser inferidos por um programa chamado motor de inferência(Reasoner). Com isso um agente autônomo pode utilizar um reasoner para inferir sobre a descrição e descobrir que serviços atendem os seus parâmetros e disponibilizam as respostas de que precisa (PRUDÊNCIO, SHEIBEL e WILLRICH, 2008). Basicamente esta inferência constrói uma hierarquia que, baseado nas propriedades de objeto e na propriedades de tipos de dados de uma classe, permite ao motor inferir se uma classe é mais especializada, mais geral, ou equivalente a outras classes. Desta forma, relações de herança e equivalências que não foram declaradas explicitamente podem ser computadas. O que ocorre também em relação aos indivíduos que, baseados em suas propriedades, pode ser inferido por um motor de inferência e este pode ser considerado uma instância de um conceito.

Diversas contribuições importantes foram feitas em QoS para WS, como por exemplo OWL-QoS, QoSOnt e Service Level Ontology (TOSIC et al., 2005). Todas estas utilizam a versão 1.0 da OWL para realizar a especificação de QoS. E com o surgimento de uma nova versão da OWL (versão 2.0), fizeram com que essas contribuições tornassem-se obsoletas. Além disso, o nível de abstração utilizado em ontologias para especificação de QoS em geral é muito alto com respeito ao domínio dos serviços web, restringindo o uso de conceitos modelados fora deste domínio.

3.3 ONTOLOGIA DE QOS PARA SISTEMAS DE COMUNICAÇÃO DE REDE

O conjunto de conceitos apresentado pelas ontologias de especificação de QoS existentes não englobam todos os conceitos necessários para serviços de rede. A maioria dos conceitos não apresentados nessas ontologias de QoS estão relacionadas a organização em camadas deste tipo de serviço. E a noção de

organização em camadas sobrepostas é essencial para modelar um sistema de comunicação, pois parâmetros em um camada podem depender de outra camada.

Alguns trabalhos mostram que a utilização de ontologias é uma abordagem promissora para o gerenciamento de QoS em sistemas de comunicação. Network Service Specification Ontology (ALÍPIO, NEVES e CARVALHO, 2007), MonONTO (MORAES et al., 2008) e SLA Ontology (GREEN apud PRUDÊNCIO, 2010) são alguns exemplos de ontologias criadas para expressar a QoS em serviços de rede. Mas estas ontologias apresentam alguns problemas como por exemplo no caso da primeira que, para permitir a criação de uma especificação de QoS mais flexível, precisaria de um conceitos mais genérico de métrica. Permitindo assim a criação de outros tipos de métricas de qualidade, envolvendo outras camadas da pilha TCP/IP. No caso da segunda, a comparação de requisitos do sistema é feita somente a nível de rede, impossibilitando fornecer uma especificação de qualidade em todos os níveis de rede, incluindo todas as camadas da pilha TCP/IP. Já no caso da terceira nota-se, de imediato, que ela não prevê a comparação de SLAs e SLSs.

Tendo esses problemas em vista e visando uma maior abrangência quanto a especificação de QoS em serviços de rede, este trabalho fará uso da ontologia NetQoSOnt (PRUDENCIO, 2010) que provê uma lista extensível de parâmetros em qualquer nível dos serviços de rede oferecendo uma maior flexibilidade em termos de parâmetros de qualidade.

3.4 NETQOSONT

Em diversas operações relacionadas ao gerenciamento da QoS, é necessário um meio eficiente de especificar QoS. A grande diversidade de soluções de QoS utilizadas pelas NSPs, cada uma com terminologia e definições próprias, torna difícil o desenvolvimento de soluções de negociação de QoS válida em todos os cenários.

Este trabalho fez uso da ontologia NetQoSOnt apresentada por Prudêncio (2010) que se propõe a ser uma ontologia para especificação de qualidade no

domínio de redes de computadores, possibilitando a criação de novas especificações e também a interpretação e comparação destas de forma automatizada.

Esta ontologia foi desenvolvida utilizando a OWL 2.0, nova versão proposta pela W3C, que supera diversas limitações das versões anteriores de OWL utilizadas pelas ontologias de QoS. E como ocorre com outras ontologias, esta continua sendo atualizada devido a necessidade de especificação de novos recursos que por ventura se fizerem necessários em novas aplicações.

A NetQoSOnt é uma ontologia de base a ser utilizada no gerenciamento de QoS oferecendo uma maior flexibilidade em termos de parâmetros de qualidade. Esta ontologia propõe classes e módulos que permitem expressar vários conceitos relacionados a especificação de QoS. Seu desenvolvimento foi influenciado pela organização em camadas dos protocolos de rede, o que permitiu uma redução da complexidade na implantação destes protocolos e na diferenciação dos serviços de rede.

Esta organização em camadas, uma sobre a outra, é importante para modelar QoS em redes de computadores, pois os parâmetros de qualidade de uma camada geralmente dependem das camadas abaixo dele, o que é particularmente importante para o mapeamento entre parâmetros independentes e dependentes de tecnologia.

Como ocorre em outras ontologias de QoS, a NetQoSOnt possui uma organização em módulos (veja figura 4), onde cada módulo é um ontologia em si e representa uma camada de rede. Reforçando a diferenciação dos recursos relacionados a cada camada, sugerindo que cada recurso tem seu lugar específico.

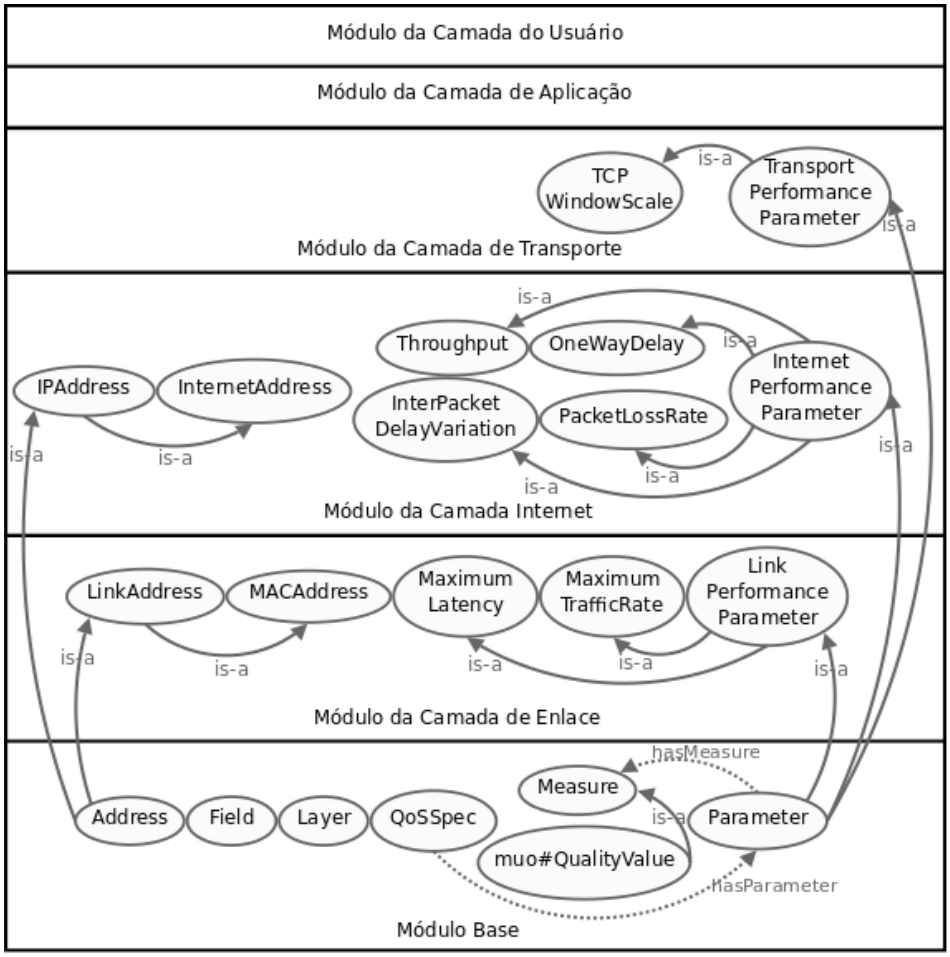

Figura 4 - Divisão em Módulos da NetQoSOnt.

O conceito de camadas é importante para modelar a QoS em redes de computadores, pois os parâmetros de qualidade de uma camada podem depender de parâmetros das camadas abaixo. Graças a esta modelagem, NetQoSOnt permite especificar o mapeamento entre parâmetros independentes de tecnologia e para parâmetros dependentes de tecnologia. A seguir são apresentados os módulos e suas características.

3.4.1 Módulo Base

O Módulo Base foi criado para englobar os conceitos genéricos para a criação de parâmetros de QoS nas mais diversas camadas. Os principais conceitos definidos neste Módulo são *Layer, Address, Field, Parameter, Measure, e QoSSpec*.

3.4.2 Módulo Da Camada De Enlace

Este Módulo agrega conceitos relacionados a camada de enlace da pilha TCP/IP. Por exemplo, a classe Address foi especializada neste Módulo para criar a classe *LinkAddress*, que representa de modo genérico os tipos de endereços existentes nesta camada.

3.4.3 Módulo Da Camada Internet

Analogamente ao Módulo da camada de enlace, o Módulo da camada Internet representa conceitos relacionados a Camada Internet da pilha TCP/IP. Address foi especializado em *InternetAddress*, criado uma classe genérica para representar endereços nesta camada, e permitindo a criação de *IPAddress*.

3.4.4 Módulo Da Camada De Transporte

Ao exemplo dos anteriores, este Módulo representa conceitos relacionados a Camada de Transporte da pilha TCP/IP. Nela a classe Parameter foi especializada em *TransportPerformaceParameter* e a classe Field foi especializada em *TransportProtocolField*. Assim como o indivíduo *TrasnportLayer* foi criado para categorizar os conceitos de Módulo.

3.4.5 Módulo Da Camada De Aplicação

Este Módulo foi designado para conter recursos pertencentes a Camada de aplicação da pilha TCP/IP, contendo especializações das classes *Parameter*, *Field* e

Address, chamadas *ApplicationPerformaceParameter, ApplicationProtocolField e ApplicationAddress* respectivamente.

3.4.6 Módulo Da Camada De Usuário

Este ultimo Módulo presente em NetQoSOnt, contém recursos para a criação de QoE (*Quality of Experience*). Neste Módulo, tendo em vista que QoE não é associada com o protocolo de aplicação ou Internet, e sim a impressão que o usuário tem durante a utilização desses, somente a classe Parameter foi especializada.

3.5 NEGOCIAÇÃO DE SLA UTILIZANDO NETQOSONT

Para abordar a transparência dos parâmetros de QoS durante a negociação do serviço de rede, esta secção demonstra como parâmetros clássicos e também novos parâmetros de QoS podem ser descritos utilizando a ontologia NetQoSOnt, e como um serviço de negociação de SLA pode levar em consideração esses parâmetros (PRUDÊNCIO, SHEIBEL e WILLRICH, 2008).

3.5.1 Uma Ontologia De SLA

A figura 5 mostra os dois principais conceitos de uma ontologia de SLA utilizando a NetQoSOnt:

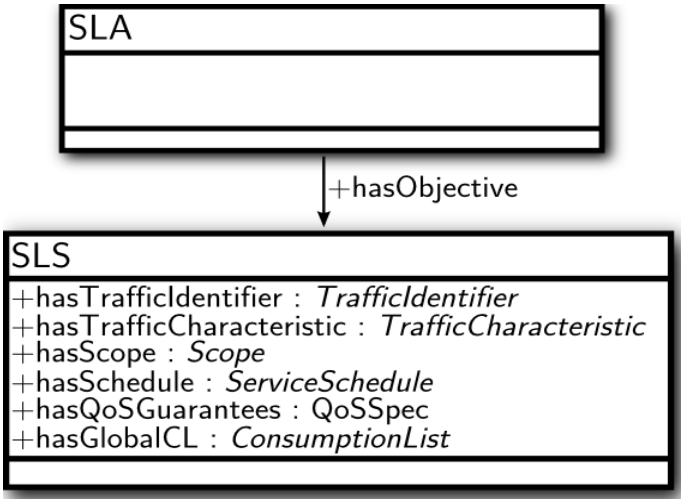

Figura 5 - Alguns conceitos de uma ontologia de SLA

Uma SLA inclui um ou mais objetivos de serviço (*hasObjective),* especificados por SLSs; e SLSs que especificam as condições sob quais os serviço deve ser entregue. O que inclui a identificação de tráfego (*hasTrafficIdentifier)* é a especificação das características do tráfego (*hasTrafficCharacteristic),* incluindo a conformidade do tráfego e a vazão . Além disso, a SLS especifica o tempo em que o serviço estará disponível para o cliente (*hasSchedule),* o escopo (*hasScope)* identificando os endereço de origem e destino em qual o serviço estará disponível e também define as características de QoS a serem garantidas através de outra SLS *QoSSpec.*

3.5.2 Transparência Na Especificação De QoS

Para ilustrar a transparência de parâmetros de QoS suportada pela NetQoSOnt, considere o cenário de negociação de serviço apresentado na Figura 6. Neste cenário um cliente ou usuário - *client* - negocia uma SLS de um serviço VoIP (Voz sobre IP). Considere também que o provedor do serviço - NSP1 – adota uma solução de QoS onde o tráfego é classificado em um conjunto de QoS, nomeados “CoS A” e “CoS B”. Neste cenário, um usuário ou sua aplicação VoIP expressa a qualidade utilizando *Mean Opinion Score(MOS)* e define que MOS deve ser maior que 4. O problema aqui ilustrado é: como o provedor do serviço automaticamente identifica, em termos de parâmetros de QoS, o que a qualidade MOS 4 significa e

como identificar qual a CoS que provê a qualidade requerida? A solução, utilizando NetQoSOnt é descrita na próxima secção.

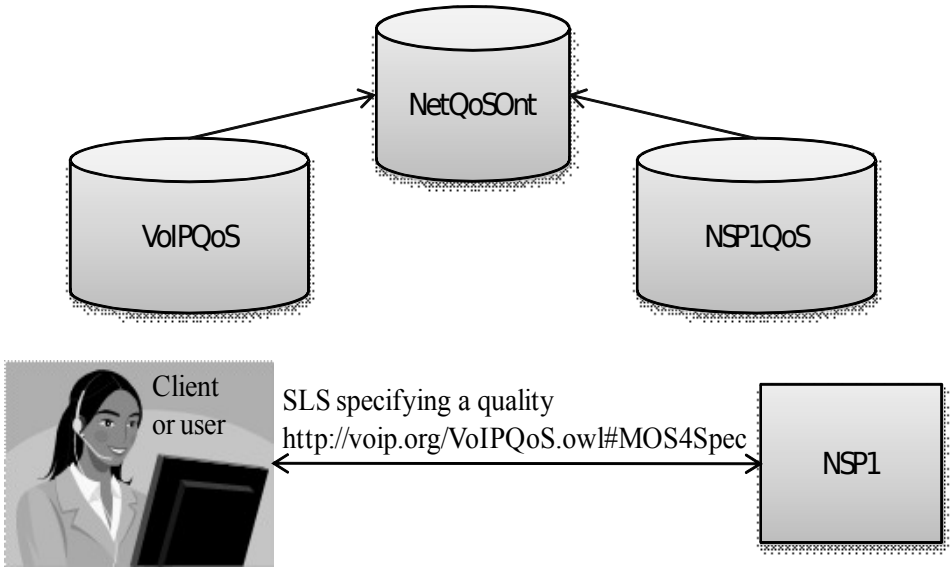

Figura 6 - Exemplo de negociação de SLA utilizando NetQoSOnt

3.5.3 Publicação Da Nova Especificação De QoS

Provedores de serviços, organizações e desenvolvedores de aplicações podem publicar a forma com que eles especificam parâmetros de QoS estendendo NetQoSOnt. Utilizando essas descrições publicadas, o provedor do serviço pode "entender" os novos parâmetros de QoS e mapeá-los em parâmetros de QoS tratados pelo seu sistema de negociação.

O cenário ilustrado na figura 6, NetQoSOnt foi utilizada por uma organização ou desenvolvedor para descrever um conjunto de especificações e parâmetros qualitativos de QoS, chamado VoIPQoS. Além disso, está organização pode descrever alguns padrões de especificação de QoS. Ela pode definir o conceito MOS4Spec, que representa uma qualidade VoIP correspondente a MOS 4. Desta forma o usuário pode qualificar o tipo de serviço através da VoIPQoS e também expressar a qualidade VoIP desejada através de MOS4Spec (que pode ser exposto

ao cliente por um metadado que represente este conceito) (PRUDÊNCIO, SHEIBEL e WILLRICH, 2008) .

3.5.4 Especificação Da QoS

A figura 6 apresenta um cliente negociando a qualidade para o seu serviço VoIP e este está utilizada o parâmetro MOS para expressar a qualidade desejada. A referência completa para a QoS desejada deve incluir a URL identificando a extensão de NetQoSOnt que define este conceito (http://voip.org/VoIPQoS.owl#MOS4Spec).

A comparação dos conceitos é feita pelo *reasoner* através de um processo *bottom-up* e este pode inferir, por exemplo, que CoS A é uma classe de serviço mais especializada que MOS4Spec. Sendo assim, através da hierarquia de inferência, deduz-se que a CoS A satisfaz os requisitos do cliente. Da mesma forma, CoS B pode ser inferida mais geral que MOS4Spec, e por isso não satisfaz os requisitos do cliente (PRUDÊNCIO, SHEIBEL e WILLRICH, 2008) .

3.5.5 Comparando As Especificações De QoS

Tendo em vista que o sistema de negociação não “conhece” o significado de MOS4Spec, este novo conceito pode ser automaticamente interpretado e, por isso, comparado com a CoS suportada pelo seu provedor de serviço.

4 SISTEMA DE GERÊNCIA DE QOS

Um sistema de gerência de QoS (também chamado de *Bandwidth Broker* ou Gerenciador de Recursos) realiza o gerenciamento de uso dos serviços com QoS em um domínio DiffServ. Como pode ser visto na figura 7, este sistema e composto de quatro módulos, módulo de negociação de SLA/SLS, módulo AAA (Autenticação, Autorização e Contabilidade), módulo de Inferência e módulo Gerenciador NETCONF, que interagem com os atores envolvidos no gerenciamento de recursos como QoS (WILLRICH et al., 2010).

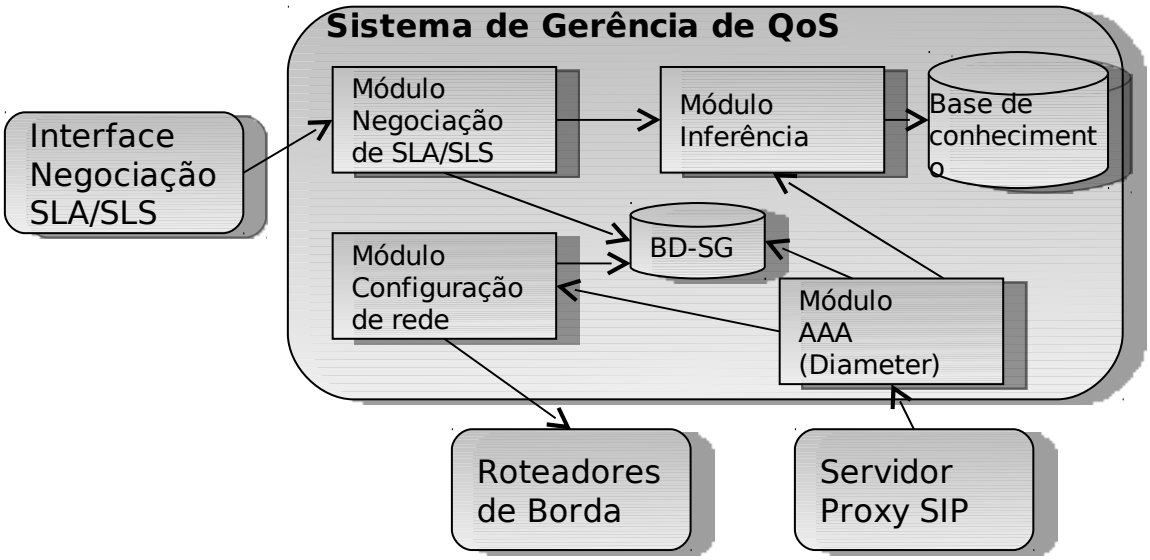

Figura 7 - Sistemas de Gerência de QoS

4.1 MÓDULO DE NEGOCIAÇÃO DE SLA/SLS

Este módulo é o lado servidor de uma interface Web que permite ao cliente da NSP negociar SLA/SLS. Esta interface tem o intuito de permitir que o cliente especifique, de maneira implícita, uma SLA, assim como as SLSs que a compõe, usando uma ontologia de SLA que reusa a NetQoSOnt. Com o uso da ontologia NetQoSOnt para especificar a QoS o cliente da NSP pode expressar suas necessidades em termos de parâmetros de QoE, por exemplo, expressar que as chamadas VoIP para um determinado destino devem ter uma qualidade MOS>4.5.

Esta interface também deve permitir a definição de perfis de usuários e o cadastramento de usuários habilitados a utilizar os serviços. A base de dados BD-SG mantém informações necessárias ao gerenciamento de alguns serviços do sistema. Em particular, ele deve manter informações dos usuários autorizados a utilizar o serviço, informações das sessões SIP ativas, entre outros.

Desta forma esta interface deve permitir que o usuário crie novas SLAs, defina as SLSs que a compõe e, obviamente, deve permitir que o serviço seja entregue com a qualidade definida.

4.2 MÓDULO DE INFERÊNCIA

Este módulo é responsável pelo processo de inferência acerca das informações encontradas em uma base de conhecimento. Esta base de conhecimento mantém as seguintes informações :

• Ontologia do provedor: contém a definição das classes de serviço do provedor e as especificações de QoS dessas classes que definem as garantias quanto o atraso, taxa de perda de pacotes, variação de atraso. Todas estas são modeladas como classes QoSSpec da ontologia NetQoSOnt;

• Ontologias de SLAs/SLSs: geradas pelo módulo de negociação de SLA/SLS.

• Subclasses de QoSSpec geradas pela negociação de sessões SIP, quando da negociação de QoS durante o estabelecimento de sessões.

• Conceitos representando grupos de usuários e usuários autorizados a utilizar o serviço.

O módulo de Inferência atua em dois momentos:

• Na negociação de uma SLS: neste momento o módulo de negociação de SLA/SLS verifica se existe alguma CoS que atende a qualidade desejada pelo cliente. Por exemplo, o usuário pode solicitar uma qualidade MOS > 4.5 para um determinado tráfego, o módulo Gestão de Conhecimento pode ser solicitado para verificar se existe alguma CoS que atenda esta qualidade para o SLS em questão.

• Na negociação da sessão SIP com QoS: o cliente pode especificar a qualidade usando parâmetros de alto nível. Neste caso o módulo de inferência usado para verificar se existe uma SLS negociada que atende a solicitação do usuário.

Este módulo também realiza uma cache que guarda na base de conhecimento as conclusões obtidas sobre novas especificações de QoS que sejam submetidas pelo sistema de gerência. Sendo assim, toda vez que e realizada uma nova negociação, essa cache e consultada para verificar se aquela especificação de QoS já existe. Caso exista, não e necessário um novo processo de inferência, pois a hierarquia já existente pode ser utilizada para descobrir se a qualidade requisitada e atendida.

4.3 MÓDULO AAA

O módulo AAA (do inglês *Authentication*, *Authorization* e *Accouting*) é o lado servidor do protocolo Diameter usado pelo servidor Proxy SIP para os procedimentos de autenticação, autorização, instalação e liberação da QoS. Este módulo interage com o módulo de inferência para tomada de decisão durante os processos de autorização e instalação da QoS. No momento da autorização da chamada SIP com QoS, o módulo de negociação deve verificar se existe um CoS que atenda a solicitação do usuário. Em seguida, ele deve verificar se esta chamada se enquadra em uma das SLSs negociadas pelo cliente.

Por exemplo, caso o usuário solicite uma qualidade MOS≥4.5, é necessário determinar se existe alguma CoS que atenda a qualidade desejada. Além de determinar a CoS que atende a solicitação do usuário, o módulo de negociação deveria verificar, entre outros, a disponibilidade de recursos de rede para atender a solicitação do cliente.

4.4 MÓDULO GERENCIADOR NETCONF

Este módulo atua como um gerente NETCONF, responsável pelo gerenciamento dos roteadores de borda da rede DiffServ. Em particular, ele é responsável pela alteração das regras de configuração dos roteadores de borda quando da instalação da QoS para uma chamada SIP com QoS. Também é responsável pela remoção da regra de classificação ao final da sessão. Estas duas operações são solicitadas pelo módulo AAA. No momento da instalação da QOS, o módulo AAA informa ao gerenciador de NETCONF a QoS desejada pela sessão SIP. Esta especificação de QoS pode ser feita usando parâmetros de QoE ou outros. Realizando consultas ao módulo de inferência, é possível realizar o mapeamento da especificação de QoS de alto nível em uma das classes de serviço da NSP.

4.5 PARÂMETROS DE AUTORIZAÇÃO DE SERVIÇO

Durante as invocações explícitas de serviços com QoS, o serviço de autorização deve receber pelo menos: (i) a identificação do usuário, (ii) a identificação do serviço, (iii) os endereços de origem e destino e (iv) o nível de QoS requisitado. As identificações de usuário e de serviço (i) e (ii) são usadas para verificar se o usuário tem o direito de invocar o serviço requisitado. Os endereços de origem e destino (iii) são usados para verificar se o escopo do serviço requisitado se encaixa em algum escopo contratado para esse serviço. Finalmente, o nível de QoS requisitado (iv) é usado para verificar se ele não é mais alto do que o nível de QoS que o usuário está autorizado a solicitar. Por exemplo, se um usuário do cliente ABC solicitar uma

chamada VoIP para a fábrica remota da mesma empresa, o servidor AAA deve saber qual o usuário está invocando o serviço (para identificar seu grupo e como consequência o seu perfil), saber qual a qualidade desejada (por exemplo, QoS Silver), e o destino da chamada (o endereço IP da rede local da fábrica remota).

4.5.1 Informações Necessárias Pelo Processo De Autorização De Serviço

O serviço de autorização deve também ter acesso: (i) ao repositório de políticas de autorização do NSP; (ii) aos perfis de usuários, que mostram quais serviços os usuários podem solicitar, seus escopos e limites de consumo; e (iii) à contabilidade do consumo dos serviços, em bases individuais, grupais e globais.

O acesso ao repositório de políticas (i) é necessário para considerar as regras de políticas que o próprio provedor pode estabelecer, que podem indicar, por exemplo, a prioridade de alguns serviços quando a carga da rede estiver acima de um determinado limiar. O acesso aos perfis de usuários (ii) autorizados pelos clientes é necessário para verificar, no momento da autorização de uma requisição de serviço, se a requisição está de acordo com o perfil do usuário, não violando nenhuma de suas restrições. O acesso à contabilidade de consumo dos serviços (iii) é necessário para implementar as validações quanto aos limites de consumo do serviço solicitado impostos para o usuário que está requisitando o serviço, para o grupo ao qual ele faz parte, e para o consumo agregado de todos os usuários do mesmo cliente.

5 PROTÓTIPO DE NEGOCIAÇÃO DE SLA/SLS

O objetivo desta seção e apresentar o protótipo para uma aplicação web que, inspirado nos avanços dos serviços web e nos conceitos de QoS e Web semântica apresentados, faz uso da ontologia NetQoSOnt para implementar o módulo de negociação de SLA/SLS e o módulo de Inferência.

Este protótipo, por motivos de simplificação e por não influenciar no resultado da solução proposta, não levou em consideração aspectos Legais, como por exemplo as responsabilidades das partes envolvidas (NSP e cliente) e o valor de eventuais multas em caso de mau uso do serviço, na criação da SLA, mantendo-se o foco em como criar uma ontologia de SLA e compor esta com suas SLSs.

5.1 PROTÓTIPO

O protótipo foi desenvolvido utilizando a linguagem JAVA e utilizou a interface de programação de aplicações OWL API(para manipulação de ontologias), juntamento com a NetQoSOnt (ontologia base para o sistema) para a implementação dos Módulo de Negociação de SLA/SLS e o Módulo de Inferência.

Este protótipo é composto por um conjunto de interfaces para a interação com o cliente, um conjunto de classes para criação e manipulação das ontologias e é exposto da seguinte forma:

1. Uma interface de autenticação do usuário (veja Figura 8), onde o cliente tem a opção de autenticar-se e/ou criar novos usuários:

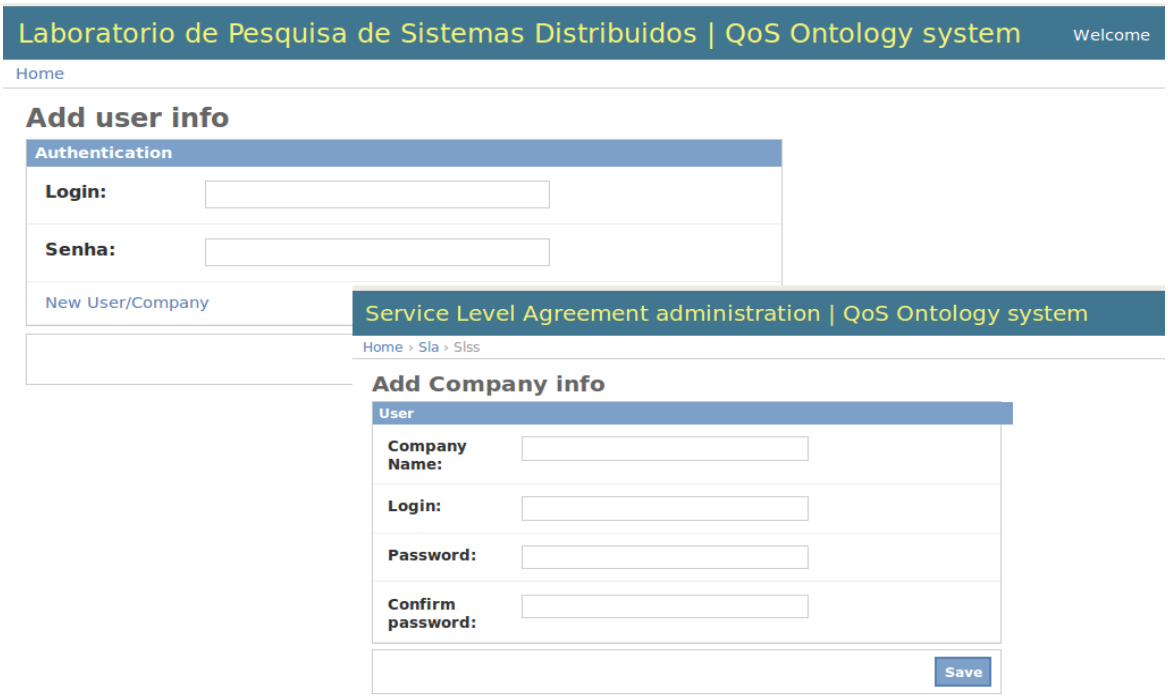

Figura 8 - Interfaces de Autenticação e Cadastro de novo usuário.

2. Uma interface para a criação de uma SLA (veja Figura 9):

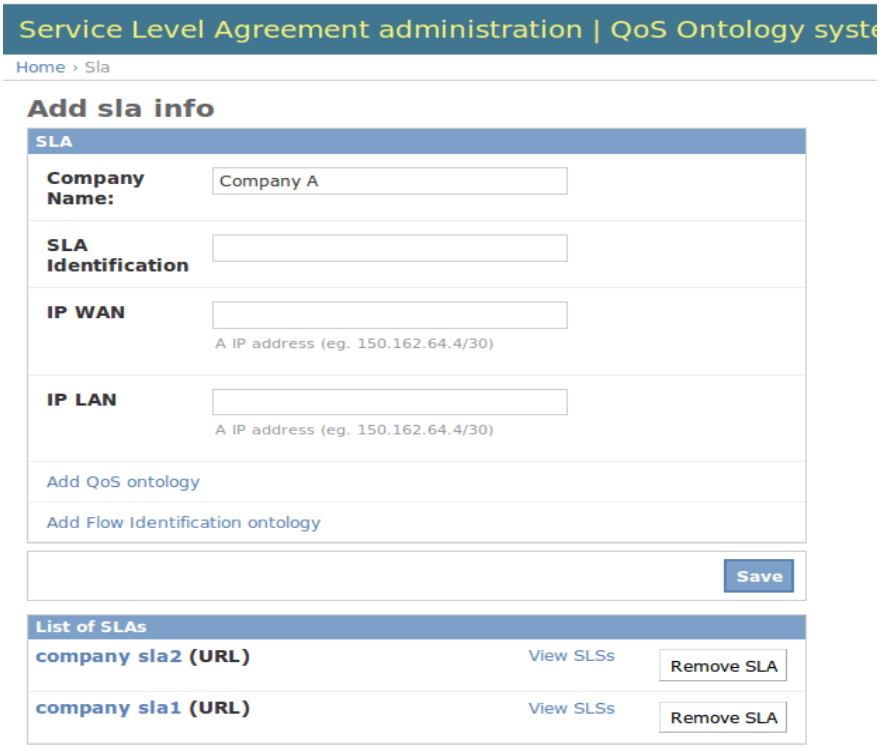

Figura 9 – Interface para criação da SLA

Nesta interface o usuário tem opção de criar uma nova SLA identificando-a em “SLA Identification” e adicionando dois endereços IP. O “IP WAN”, que seria o endereço IP externo e o “IP LAN” que seria o IP interno da rede. Considera-se aqui que a máquina do cliente encontra-se dentro de uma rede privada (IP LAN) e possui acesso a uma rede externa (IP WAN). Veja na figura 10 como são representados os endereços IPs na ontologia desta SLA.

```
<owl:Class rdf:about="&CompanyA-SLA;CompanyA-SLA">
   <rdfs:subClassOf rdf:resource="#SLA"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#hasIPWAN"/>
         <owl:hasValue rdf:resource="&CompanyA-SLA;CompanyA-IPWAN"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#hasIPLAN"/>
         <owl:hasValue rdf:resource="&CompanyA-SLA;CompanyA-IPLAN"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   ...
   ...
   ...
 </owl:Class>

 <NetOnt:IPAddress rdf:about="&CompanyA-SLA;CompanyA-IPWAN">
   <rdf:type rdf:resource="&owl;Thing"/>
   <BaseOnt:hasIdentifier rdf:datatype="&xsd;string">150.150.64.100</BaseOnt:hasIdentifier>
</NetOnt:IPAddress>

<NetOnt:IPAddress rdf:about="&CompanyA-SLA;CompanyA-IPLAN">
   <rdf:type rdf:resource="&owl;Thing"/>
   <BaseOnt:hasIdentifier rdf:datatype="&xsd;string">192.168.0.20</BaseOnt:hasIdentifier>
</NetOnt:IPAddress>
```

Figura 10 – Representação do IP WAN e IP LAN na ontologia.

A implementação da classe responsável pelas ações mencionadas acima é a MakeSLAOntology.java (anexo A).

Na interface apresentada na figura 10 aparece também dois *links* - *“*Add QoS ontology” e “Add Flow Identification ontology” - que direciona o usuário a uma nova interface aonde este pode adicionar ontologias de QoS e de identificação de fluxo

para compor a base de conhecimento desta empresa (veja figura 11). Essas ontologias serão utilizadas posteriormente na definição das SLSs desta SLA.

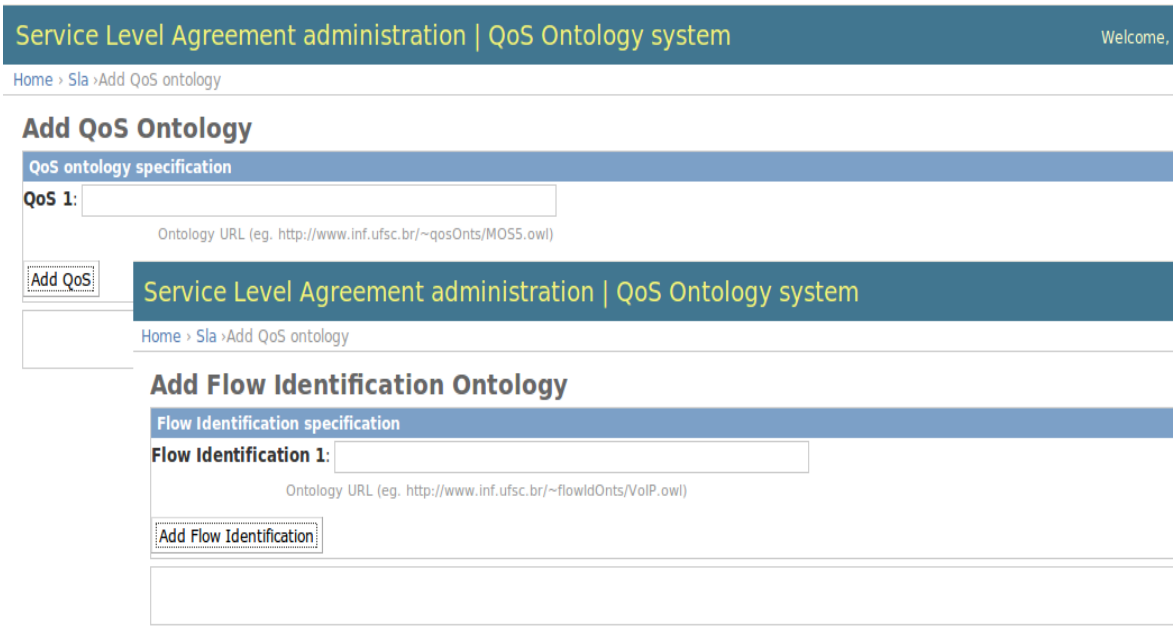

Figura 11 – Interfaces para inserir as ontologias de QoS e Identificação de fluxo

A implementação das classes responsáveis pela ações mencionadas são *SaveQosOntology.java* e *SaveFlowOntology* respectivamente (veja anexo A).

3. Interfaces para para definir as SLSs que irão compor uma SLA (Figura 12):

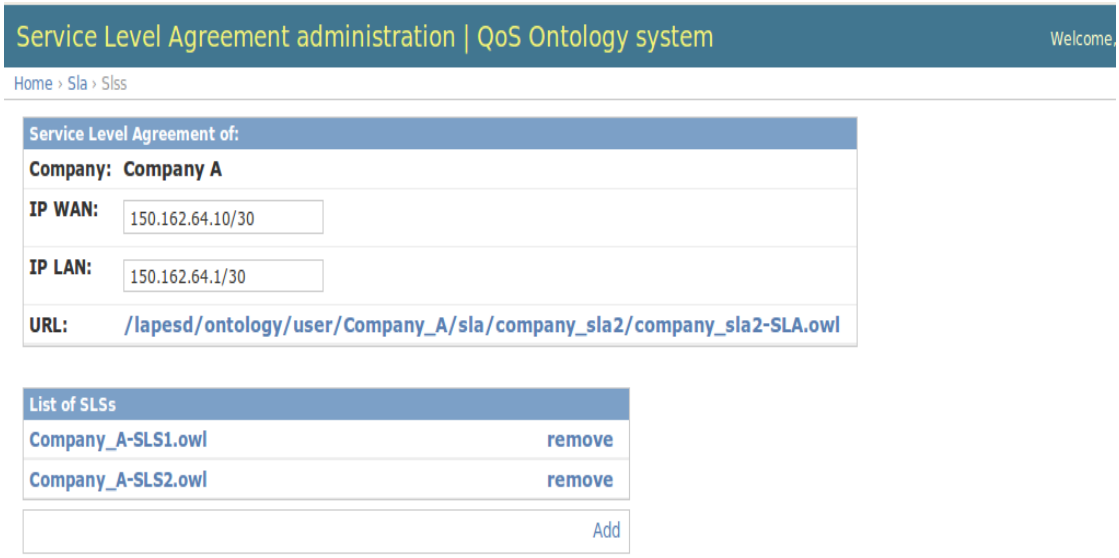

Figura 12– Interface para adicionar as SLSs que irão compor uma SLA

Uma vez criada uma SLA é necessário definir as especificações de serviço (SLS) que irão compor a mesma. Na Figura 12 é apresentada a SLA em questão, no caso “comapny_sla2-SLA.owl”, e suas SLSs (e.g Company_A-SLS1.owl).

Para adicionar uma SLS o usuário deve clicar em “Add” o que o levará a uma nova interface (veja Figura 13) aonde ele pode optar por adicionar uma SLS dinâmica (do inglês *Dynamic SLS)* ou uma SLS estatica (do inglês *Static SLS)*.

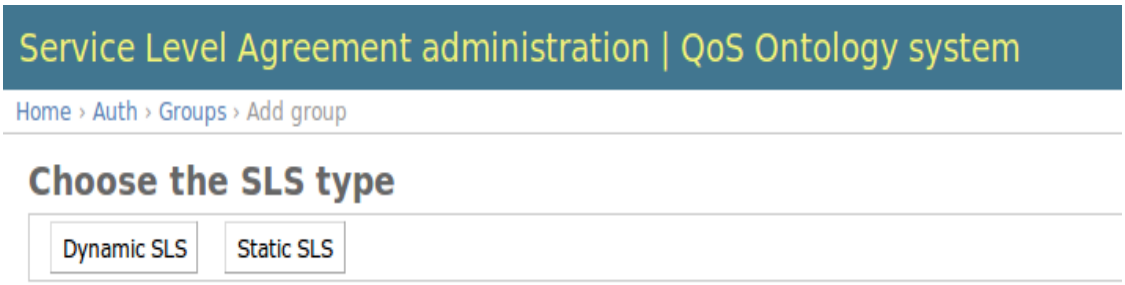

Figura 13 – Interface para definição do tipo da SLS

Ao optar por uma SLS dinâmica (do inglês *Dynamic SLS)* o usuário é direcionado para uma nova interface (veja Figura 14) aonde poderá definir o escopo (do inglês *Scope)* do serviço, a Identificação do Fluxo (do inglês *Flow Identification*), através da escolha de uma das ontologias que foram adicionadas anteriormente (veja figura 11) a base de conhecimento, e também, desta mesma forma, é definida a especificação da Qualidade de Serviço (do inglês *QoS Specification*) desejada.

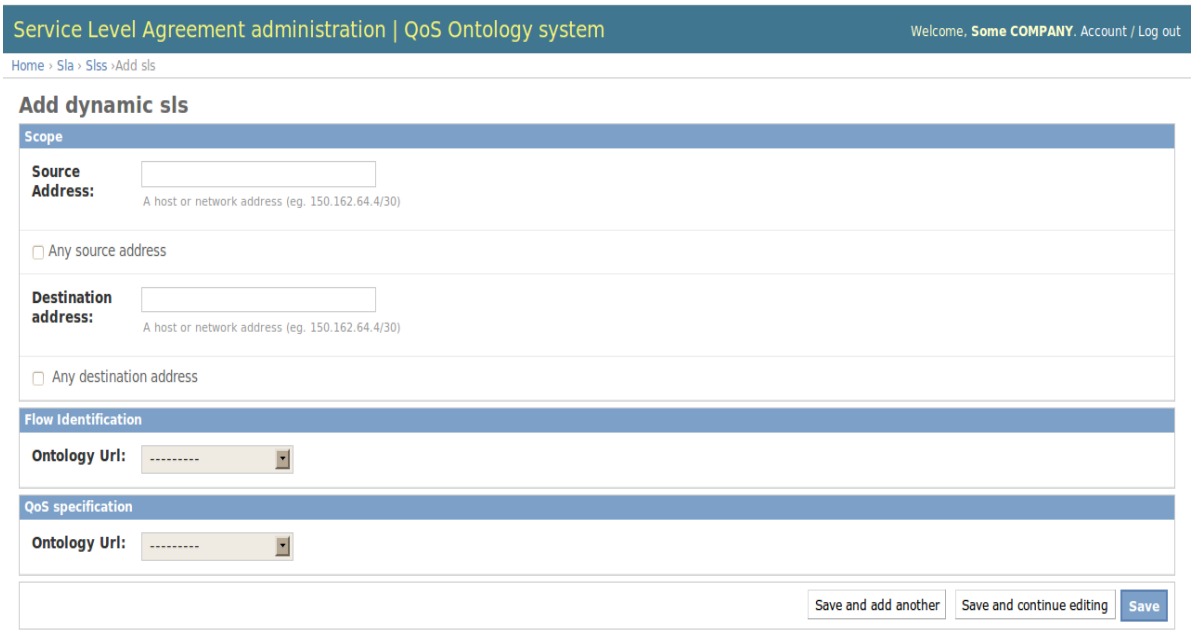

Figura 14 – Interface para definição da SLS dinâmica

Para cada SLS definida será criada uma ontologia da mesma e esta ontologia, após ser inferida pelo motor de inferência, será salva na base de conhecimento e irá compor uma SLA. Na Figura 15 pode ser observado como essa relação é expressada na ontologia.

```
<owl:Class rdf:about="&CompanyA-SLA;CompanyA-SLA">
   <rdfs:subClassOf rdf:resource="#SLA"/>
   ...
   ...
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#hasSLS"/>
         <owl:someValuesFrom rdf:resource="&CompanyA-SLS;CompanyA-SLS2"/>
      </owl:Restriction>
   </rdfs:subClassOf>
   ...
   ...
 </owl:Class>

<owl:Class rdf:about="&CompanyA-SLS;CompanyA-SLS2">
     <rdfs:subClassOf rdf:resource="#SLS"/>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="&TransOnt;hasFlowID"/>
             <owl:hasValue rdf:resource="&CompanyA-SLS;CompanyA-FlowID"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="#hasSpecification"/>
             <owl:someValuesFrom rdf:resource="&CompanyA-SLS;CompanyA-Specification1"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="&TransOnt;hasScope"/>
             <owl:hasValue rdf:resource="&CompanyA-SLS;CompanyA-Scope"/>
         </owl:Restriction>
     </rdfs:subClassOf>
 </owl:Class>

<owl:Thing rdf:about="&CompanyA-SLS;CompanyA-Scope">
     <rdf:type rdf:resource="&TransOnt;Scope"/>
     <TransOnt:hasDestinationPort rdf:resource="&CompanyA-SLS;10"/>
     <TransOnt:hasSourcePort rdf:resource="&CompanyA-SLS;30"/>
 </owl:Thing>

<NetOnt:DestinationIPAddress rdf:about="&CompanyA-SLS;dst_address:150.162.66.4/30">
     <rdf:type rdf:resource="&owl;Thing"/>
     <BaseOnt:hasIdentifier rdf:datatype="&xsd;string">150.162.66.4/30</BaseOnt:hasIdentifier>
 </NetOnt:DestinationIPAddress>
```

Figura 15 – Composição de uma SLA com o escopo e as ontologias que formam a SLS

A implementação da classe responsável pelas ações citadas anteriormente é a *MakeSLSOntology.java (*veja anexo A).

5.1.2 Motor De Inferência

Uma das grandes vantagens do uso de ontologias está no uso de motores de inferência (PRUDÊNCIO, 2010). Com eles, é possível assegurar a validade dos conceitos e ralações criadas em um arquivo OWL. O uso incorreto da sintaxe e da semântica das construções de OWL geram uma inconsistência, que pode sem capturadas pelo motor de inferência e geralmente constituem erro de modelagem que devem ser concertados.

A implementação do motor de inferência pode ser vista em *InferenceServer.java* (veja anexo A), e este é responsável por inferir sobre a ontologia de SLA/SLS criada e a ontologia do provedor do serviço, composta pelas classes de serviço que são fornecidas por ele. O motor de inferência recebe essas duas ontologias como parâmetros, infere sobre elas, e retorna as classes de serviço da NSP que atendem a requisição do cliente. Por exemplo, um cliente pode requisitar um serviço com qualidade MOS4 e após a inferência conclui-se que, para aquele provedor, as classes de serviço que atendem a QoS exigida são AF11 (do inglês *Assured Forwarding*) e EF(do inglês *Expedited Forwarding*). Aqui cabe ao provedor do serviço estipular o custo para cada classe. Pode por exemplo, estipular que para utilizar a classe EF, que é a CoS de maior prioridade, o cliente pagaria mais pelo serviço ou através de perfis de usuário o provedor pode estipular qual será a classe utilizada.

5.2 TESTES

A fim de testar a solução proposta, em termos funcionais e de desempenho, foram realizados testes em uma estrutura de rede formada por roteadores DiffServ Linux. Para esta experimentação, parte das funcionalidades da solução proposta foi implementada. O objetivo não foi o de implementar todas as funcionalidades dos protocolos NETCONF e Diameter utilizados na soluções. A meta foi avaliar a eficiência do uso de uma abordagem semântica para especificar a QoS e validar o mapeamento desta especificação em parâmetros de configuração de rede.

A Fig. 16 apresenta a estrutura de deste implantada. Ela é composta dois roteadores DiffServ Linux, dois agentes usuários SIP, um sistema de gerência de QoS e um servidor Proxy SIP.

Em relação aos agentes usuários, foram utilizados dois softwares, ferramenta SIP Inspector e o softphone X-Lite. O SIP Inspector foi utilizado, pois ela permite a edição dos campos da mensagem INVITE, permitindo a inclusão do atributo necessário à solicitação de um serviço com QoS.

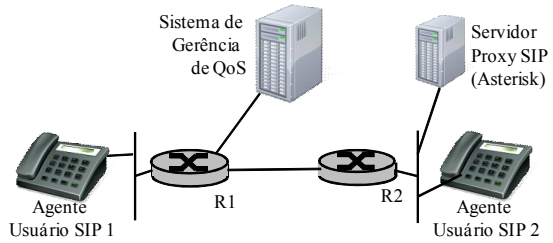

Figura 16. Estrutura de Teste

O sistema VoIP utiliza o PBX IP Asterisk, que foi estendido para interpretar o novo atributo SDP proposto. Além disso, o Asterisk foi alterado para realizar as operações de AAA utilizando a um cliente Diameter instalado na mesma máquina. Este cliente foi desenvolvido na linguagem C, e é responsável pelas solicitações de autenticação, autorização, instalação e encerramento de sessões com QoS. Ele implementa as funções mínimas necessárias à realização do teste. Este processo

Algumas das funcionalidades básicas do sistema de gerência foram prototipadas:

- O módulo de inferência foi desenvolvido em Java e utiliza a API OWL 2.0. Para este experimento, tanto a ontologia do provedor quando a utilizada para expressar a QoS pelo usuário.

- Para simular o funcionamento do gerente NETCONF foi implementado um script Python usando a biblioteca XML-RPC, com recursos mínimos para suportar RPC codificadas em XML. Este módulo simula o uso do protocolo NETCONF com

suporte a operações de edição de regras de classificação nos roteadores de borda DiffServ, bem como operação para eliminação da regras.

Em vez de serem instalados agentes NETCONF nos roteadores de borda foi executado um script Python também usando a biblioteca xmlrpc e o banco sqlite. Este sistema simula o funcionamento de um agente NETCONF sobre Linux.

Para este teste foram estabelecidas 10 sessões SIP iniciadas pelo agente usuário 1 (Usando SIP Inspector), sendo que 5 foram realizadas sem a QoS como precondição e 5 chamadas com negociação da QoS. A QoS solicitada foi a de *http://voiporg.org/#MOS4.5-Spec*, sendo que a ontologia do provedor é a mesma que apresentada na seção 2. Portanto, a CoS inferida para atender a chamada é EF. O codec utilizados para todas as chamadas foi o G.711 com tamanho de pacote de voz de 20ms.

Durante a realização das chamadas, foram realizadas capturas de tráfego no agente usuário SIP 1, usando o analisador de protocolos Wireshark. A partir dele foi possível medir diversos atrasos médios nas sinalizações das chamadas sem e com negociação da QoS.

A medida do atraso médio da solicitação da chamada SIP sem QoS até a recepção da mensagem SIP *180 Ringing* pelo agente usuário SIP 1 foi de 14ms. Para chamadas com QoS, o atraso médio foi de 578ms. Este acréscimo de atraso da sinalização é devido ao processo de sinalização da AAA, e principalmente devido a inferência na ontologia. É importante notar que esta inferência ocorrerá somente na primeira utilização da especificação de QoS por parte de todos os usuários do serviço. Nas seguintes, o resultado da inferência já estará incluído na base de conhecimento. Usando este mecanismo, o atraso de sinalização reduzirá em média para 446ms. Considera-se aqui que este incremento de aproximadamente meio segundo no estabelecimento da chamada é quase imperceptível pelos usuários do serviço.

Para avaliar o atraso da instalação da QoS após o atendimento do telefone, foi medido no computador do agente usuário SIP 1 o número de pacotes de voz que

chegaram marcados com BE antes da inclusão da nova regra de classificação dos pacotes. Foi verificado o valor de um pacote de voz em todas as medidas, que corresponde a um tempo total de voz de 20ms. Isto quer dizer que após 20ms de comunicação, o tráfego será marcado com a classe negociada. Esta medida demonstra que o usuário não deverá perceber esta não exigência momentânea da qualidade solicitada.

Para medir o impacto na qualidade da voz na troca da classificação dos pacotes de voz, foi medido o tempo entre o último pacote marcado com BE e o primeiro marcado com EF. Em condições ideais, este atraso seria na ordem de 20ms, e o valor médio medido foi de 24ms e esta variação de atraso está dentro da média da chamada.

Considerando a aplicação da solução proposta em um domínio real de rede, deve-se também considerar os atrasos de rede do domínio e os atrasos da importação das ontologias de QoS. Para o estabelecimento da sessão SIP, além do atraso de meio segundo adicional de sinalização e processamento no experimento, deve-se acrescer o atraso no acesso a ontologia externa (no exemplo, VoIPQoSOnt.owl) no momento da primeira utilização de um conceito desta ontologia por parte de qualquer um usuário do sistema. O atraso de transferência dependerá muito do tamanho da ontologia e de sua localização. No caso foram realizadas medidas de carga em algumas ontologias de porte médio e chegou-se a medidas abaixo de 6s. Este valor é considerado tolerável para o estabelecimento da sessão SIP, e ocorrerá somente na primeira chamada de qualquer usuário do sistema que utilizar um parâmetro de QoS ainda desconhecido pela NSP.

6 CONCLUSÃO

Neste capítulo serão apresentados os resultados obtidos na realização deste trabalho e as perspectivas para trabalhos futuros.

6.1 **RESULTADOS OBTIDOS**

Neste trabalho foi proposto um modelo para negociação de acordos de nível de serviço onde, durante o período de validade do contrato (ou SLA) o usuário e o provedor do serviço possam criar, modificar e remover as especificações de serviço (SLS) , oferecendo um maior dinamismo na hora de definição da qualidade de serviço e dando ao usuário um maior poder de escolha do nível de QoS desejado.

Foi utilizada uma interface de programação de aplicações denominada OWL API 2. Esta API foi desenvolvida utilizando a linguagem JAVA e sua ultima versão tem como foco a OWL 2 e esta suporta a análise e processamento de ontologias nas sintaxes definidas pela W3C.

Além disso, foi utilizada uma ontologia para a definição de especificações de QoS que oferece flexibilidade em termos de parâmetros de qualidade denominada NetQoSOnt. Com base nesta ontologia podem ser definidas soluções de QoS permitindo aos clientes e usuários de serviço de rede expressar suas necessidades em termos de QoS usando parâmetros de qualidade quantitativos e qualitativos, em todos os níveis de rede, desde a qualidade percebida a parâmetros da camada de enlace.

No modelo proposto, a comparação de parâmetros de QoS é realizada automaticamente por um motor de inferência. Ao carregar a ontologia que contém o pedido do cliente e a ontologia que contém as CoSs do provedor, o cálculo da nova hierarquização das classes permite ao provedor saber quais de suas CoSs atendem ao pedido do cliente simplesmente consultando o motor de inferência por quais de

suas CoSs que foram inferidas como subclasses ou classes equivalentes da classe que representa especificação do cliente.

6.2 TRABALHOS FUTUROS

Como trabalhos futuros a este, são apresentados os seguintes temas:

- Integrar este sistema com os módulos: Módulo AAA (do inglês *Authentication, Authorization e Accouting) e* Módulo Gerenciador NETCONF.

- Integrar ao sistema um modelo de perfis de usuário para autorização de serviços de QoS. Utilizando ontologias para a definição dos conceitos (perfis), cada perfil pode indicar os serviços que o usuário está autorizado a requisitar. Políticas de autorização baseadas em perfil de usuário podem ser definidas e executadas durante a invocação de serviços com QoS explícita.

- Aplicação da solução proposta em um ambiente real.

REFERÊNCIAS

D. Black et al. An Architecture for Differentiated Services, RFC 2475, Internet Engineering Task Force, 1998.

Royer, J., Willrich, R. e Diaz, M. (2008). “User Profile-Based Authorization Policies for Network QoS Services”. 7th IEEE Int. Symp. on Network Computing and Applications (NCA), pp. 68-75.

K. Chan, et al. COPS Usage for Policy Provisioning (COPS-PR), RFC 3084, 2001.

R. Enns. NETCONF Configuration Protocol, RFC 4741, Internet Engineering Task Force, 2006.

G. Camarillo, W. Marshall, J. Rosenberg. Integration of Resource Management and Session Initiation Protocol (SIP), RFC 3312, Internet Engineering Task Force, 2002.

J. Polk, S. Dhesikan, G. Camarillo. Quality of Service (QoS) Mechanism Selection in the Session Description Protocol (SDP), RFC 5432, Internet Engineering Task Force, 2009.

H.J. Park, J.J. Yang, J.K. Choi, H.S. Kim. QoS Negotiation for IPTV Service using SIP. 9th Int. Conf. on Advanced Communication Technology, pp. 945-948, 2007.

M. Alexander, P. Suppan. An Architecture for SDP-based Bandwidth Resouce Allocation with QoS for SIP in IEEE 802.16 Networks. 2nd Int. Workshop on Quality of Service & Sercurity for Wireless and Mobile Networks, pp. 75-82, 2006.

J. Rosenberg et al. SIP: Session Initiation Protocol, RFC 3261, Internet Engineering Task Force, 2002.

M. Handley et al. SDP: Session Description Protocol, RFC 4566, Internet Engineering Task Force, 2006.

R. Willrich, L.H. Vicente, R.B. Uriarte, A.C. Prudêncio, J. Cé Júnior. Invocação Dinâmica de Serviços com QoS em Sessões Multimídia SIP. 8th Int. Information and Telecommunication Technologies Symposium (I2TS), 2009.

A.C. Prudêncio, R. Willrich, S. Tazi , M. Diaz. Quality of Service Specifications: A Semantic Approach. In: 8th IEEE International Symposium on Network Computing and Application, pp. 219-226, 2009.

P. Calhoun, et al. Diameter Base Protocol, RFC 3588, Internet Engineering Task Force, 2003.

D. Sun et al. Diameter Quality of Service Application, RFC 5866, Internet Engineering Task Force, 2010.

The Open Source PBX & Telephony Platform. URL: http://www.asterisk.org/, 2010.

SIP Inspector. http://sourceforge.net/projects/sipinspector/, 2010.

X-Lite. http://www.counterpath.com/x-lite.html, 2010.

Wireshark. http://www.wireshark.org/, 2010.

ANEXO A – CÓDIGO FONTE

INFERENCESERVER.JAVA

```
import java.io.File;
import java.net.URI;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.inference.OWLReasonerAdapter;
import org.semanticweb.owl.model.AxiomType;
import org.semanticweb.owl.model.OWLAxiom;
import org.semanticweb.owl.model.OWLClass;
import org.semanticweb.owl.model.OWLDescription;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyCreationException;
import org.semanticweb.owl.model.OWLOntologyManager;
import org.semanticweb.owl.util.CommonBaseURIMapper;
import uk.ac.manchester.cs.factplusplus.owlapi.Reasoner;
import uk.ac.manchester.cs.owl.OWLEquivalentClassesAxiomImpl;
import uk.ac.manchester.cs.owl.OWLSubClassAxiomImpl;

/**
 * @author Felipe B. Teixeira
 */
public class InferenceServer {

  private OWLOntologyManager manager =
      OWLManager.createOWLOntologyManager();
  private String netQoSOntBaseURIStr;
  private String ontologiesLocalDir;
  private CommonBaseURIMapper mapper;
  private OWLReasonerAdapter reasoner;

  public InferenceServer() {
    this.netQoSOntBaseURIStr = "http://www.inf.ufsc.br/~achilles/mestrado/";
    this.ontologiesLocalDir =
      "/home/user/workspace/lapesd/ontologiesPrototipo/";
    this.mapper = new CommonBaseURIMapper(URI.create("file://" +
      this.ontologiesLocalDir));
    File dir = new File(this.ontologiesLocalDir);
    String localOntologyFiles[] = dir.list();
    for (String file : localOntologyFiles)
      this.mapper.addMapping(URI.create(this.netQoSOntBaseURIStr +
            file.toString()), this.ontologiesLocalDir + file.toString());
    this.manager.addURIMapper(this.mapper);
    try {
      this.reasoner = new Reasoner(this.manager);
    } catch (Exception ex) {
      Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
      null, ex);
      System.out.println("Error instatiating the reasoner");
    }
   Logger l =
      Logger.getLogger("uk.ac.manchester.cs.factplusplus.owlapi.Ontolog
```

```
            yLoader");
    l.setLevel(Level.OFF);
  }

//Returns a Hashtable<OWLClass,ArrayList<OWLClass>> containing the provider
      specifications that conform to the client specifications
  public Hashtable<OWLClass,ArrayList<OWLClass>>
      matchingProviderSpecifications(String clientOntologyURIStr,String
            providerOntologyURIStr){
    Hashtable<OWLClass,ArrayList<OWLClass>> matchingProviderSpecs = new
      Hashtable<OWLClass,ArrayList<OWLClass>>();
    try
    {
      OWLClass qosSpec = this.manager.getOWLDataFactory()
            .getOWLClass(URI.create(this.netQoSOntBaseUR   IStr +
                  "BaseOnt.owl#QoSSpec"));
      OWLOntology clientOntology = this.manager
            .loadOntologyFromPhysicalURI(URI.create(clientOntologyURIStr));
      ArrayList<Object> clientSpecs = new ArrayList<Object>();

      for(OWLAxiom axiom : clientOntology.getAxioms())
        if(axiom.getReferencedEntities().contains(qosSpec))
          if(axiom.getAxiomType().equals(AxiomType.EQUIVALENT_CLASSES))
          {
            OWLEquivalentClassesAxiomImpl aux =
                  (OWLEquivalentClassesAxiomImpl) axiom;
            clientSpecs.addAll(aux.getNamedClasses());
          }else if (axiom.getAxiomType().equals(AxiomType.SUBCLASS))
          {
            OWLSubClassAxiomImpl aux = (OWLSubClassAxiomImpl) axiom;
            clientSpecs.add(aux.getSubClass());
          }
    Set<OWLOntology> clientOntologyImportsClosure =
      this.handleOntology(clientOntology);
    Set<OWLOntology> providerOntologyImportsClosure =
      this.handleOntologyString(providerOntologyURIStr);

    this.reasoner.loadOntologies(providerOntologyImportsClosure);
    this.reasoner.loadOntologies(clientOntologyImportsClosure);

    System.out.println("Classifying ontologies (and measuring Reasoner
      response time)");
    double start = System.currentTimeMillis();
    this.reasoner.classify();
    double elapsedTime = System.currentTimeMillis() - start;
    System.out.println("done.");
    System.out.println("Cassified in" + elapsedTime + "ms");

    OWLClass owlNothing = this.manager.getOWLDataFactory()
      .getOWLClass(URI.create("http://www.w3.org/2002/07/owl#Nothing"));

    ArrayList providerSpecs;
    for(Object clientSpec : clientSpecs)
    {
      providerSpecs = new ArrayList();
      providerSpecs.addAll(OWLReasonerAdapter.flattenSetOfSets(this.reasoner
            .getDescendantClasses((OWLDescription) clientSpec)));
      providerSpecs.addAll(this.reasoner.getEquivalentClasses((OWLDescriptio
            n)clientSpec));
      providerSpecs.removeAll(clientSpecs);
      providerSpecs.remove(owlNothing);
      matchingProviderSpecs.put((OWLClass) clientSpec,providerSpecs);
    }
```

```
    } catch (OWLOntologyCreationException ex) {
      Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
            null, ex);
    } catch (Exception ex) {
      Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
            null, ex);
    }
    return matchingProviderSpecs;
  }

  public Set<OWLOntology> handleOntology(OWLOntology ontology) throws
      Exception {
    Set<OWLOntology> ontologyImportsClosure =
      this.manager.getImportsClosure(ontology);
    if( !ontologyImportsClosure
      .contains(this.manager.getOntology(URI.create(this.netQoSOntBaseURIStr
            + "BaseOnt.owl"))))
      throw new Exception("Not NetQoSOntDerivedException " +
            ontology.getURI().toString());
  return ontologyImportsClosure;
  }

  public Set<OWLOntology> handleOntologyString(String ontologyString){
    OWLOntology ontology = null;
    Set<OWLOntology> ontologyImportsClosure = null;
    try {
      ontology= this.manager
            .loadOntologyFromPhysicalURI(URI.create(ontologyString));
      ontologyImportsClosure = this.handleOntology(ontology);
    } catch (OWLOntologyCreationException ex) {
      Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
            null, ex);
    } catch (Exception ex) {
      Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
            null, ex);
    }
    return ontologyImportsClosure;
  }
}
```

SAVEFLOWONTOLOGY.JAVA

```
import java.io.IOException;
import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.io.RDFXMLOntologyFormat;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyCreationException;
import org.semanticweb.owl.model.OWLOntologyManager;
import org.semanticweb.owl.model.OWLOntologyStorageException;
import org.semanticweb.owl.model.UnknownOWLOntologyException;
```

```
/**
 * @author Felipe B. Teixeira
 */
public class SaveFlowOntology extends HttpServlet {

    private OWLOntologyManager manager;

    protected void processRequest(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
        String flow = "flow";
        String flowOnt = null;
        int cont = 1;
        while(true){
          flowOnt = request.getParameter(flow + cont);
          if(flowOnt == null || flowOnt.equals(""))
            break;
          else{
            cont++;
            this.saveQoSOntology(flowOnt);
          }
        }
        RequestDispatcher rd
            = request.getRequestDispatcher("/jsp/add/AddSlaInfo.jsp");
        rd.forward(request, response);
    }

    private void saveQoSOntology(String url){

      String[] aux = url.split("/");
      System.out.println(aux[aux.length - 1]);

      this.manager = OWLManager.createOWLOntologyManager();
      OWLOntology ontology = null;
      try {
        ontology = this.manager.loadOntology(URI.create(url));
      } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(SaveQosOntology.class.getName()).log(Level.SEVERE,
            null, ex);
      }

      URI physicalURI = URI.create("file:" +
      this.getServletContext().getRealPath("/") + "ontology/user/" +
      "Company_A" + "/flow_ontology/" + aux[aux.length -1]);
       try {
            this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
                  physicalURI);
       } catch (OWLOntologyStorageException ex) {
       Logger.getLogger(SaveQosOntology.class.getName())
            .log(Level.SEVERE, null,ex);
    } catch (UnknownOWLOntologyException ex) {
      Logger.getLogger(SaveQosOntology.class.getName()).log(Level.SEVERE,
            null, ex);
    }
    }
}
```

SAVEQOSONTOLOGY.JAVA

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.io.RDFXMLOntologyFormat;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyCreationException;
import org.semanticweb.owl.model.OWLOntologyManager;
import org.semanticweb.owl.model.OWLOntologyStorageException;
import org.semanticweb.owl.model.UnknownOWLOntologyException;

/**
  * @author Felipe B. Teixeira
 */
public class SaveQosOntology extends HttpServlet {

    private OWLOntologyManager manager;

    protected void processRequest(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String qos = "qos";
            String qosOnt = null;
            int cont = 1;
            while(true){
              qosOnt = request.getParameter(qos + cont);
              if(qosOnt == null || qosOnt.equals(""))
                break;
              else{
                cont++;
                this.saveQoSOntology(qosOnt);
              }
            }
            RequestDispatcher rd = request
            .getRequestDispatcher("/jsp/add/AddSlaInfo.jsp");
            rd.forward(request, response);
        } finally {
            out.close();
        }
    }

    private void saveQoSOntology(String url){

      String[] aux = url.split("/");
      System.out.println(aux[aux.length - 1]);

      this.manager = OWLManager.createOWLOntologyManager();
      OWLOntology ontology = null;
      try {
        ontology = this.manager.loadOntology(URI.create(url));
```

```
      } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(SaveQosOntology.class.getName()).log(Level.SEVERE,
            null, ex);
      }

      URI physicalURI = URI.create("file:" + this.getServletContext()
            .getRealPath("/") + "ontology/user/" + "Company_A" +
            "/qos_ontology/" + aux[aux.length -1]);
    try {
      this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
                  physicalURI);
    } catch (OWLOntologyStorageException ex) {
      Logger.getLogger(SaveQosOntology.class.getName()).log(Level.SEVERE,
            null, ex);
    } catch (UnknownOWLOntologyException ex) {
      Logger.getLogger(SaveQosOntology.class.getName()).log(Level.SEVERE,
            null, ex);
    }
    }
}
```

MAKESLAONTOLOGY.JAVA

```
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.semanticweb.owl.util.SimpleURIMapper;
import org.semanticweb.owl.io.RDFXMLOntologyFormat;
import java.net.URI;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.servlet.RequestDispatcher;
import org.semanticweb.owl.model.*;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.util.OWLOntologyWalker;

/**
 * @author Felipe B. Teixeira
 */
public class MakeSLAOntology extends HttpServlet {

  private String slaName, companyName;
  private String borderRouter;
  private URI base;
  private OWLOntologyManager manager;
  private OWLOntologyWalker walker;
  private VisitorSLA visitor;
  private List<OWLOntology> importedOntologies;

  protected void processRequest(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
    try {
      this.companyName = request.getParameter("company_name").replaceAll("
            ", "_");
```

```
this.slaName = request.getParameter("sla_name").replaceAll(" ", "_");
this.borderRouter = request.getParameter("border_router");

//Cria diretorio com o nome da empresa para armazenar as slas e suas
slss
new File(this.getServletContext().getRealPath("/") + "ontology/user/"
      + this.companyName).mkdir();
new File(this.getServletContext().getRealPath("/") + "ontology/user/"
      + this.companyName + "/qos_ontology").mkdir();
new File(this.getServletContext().getRealPath("/") + "ontology/user/"
      + this.companyName + "/flow_ontology").mkdir();
java.io.File companyFile = new
      File(this.getServletContext()
      .getRealPath("/") + "ontology/user/" +   this.companyName);
companyFile.mkdirs();
boolean success = (new File(this.getServletContext().getRealPath("/")
      + "ontology/user/" + this.companyName + "/sla")).mkdir();
success = (new File(this.getServletContext().getRealPath("/") +
      "ontology/user/" + this.companyName + "/sla/" +
            this.slaName)).mkdir();
success = (new File(this.getServletContext().getRealPath("/") +
      "ontology/user/" + this.companyName + "/sla/" + this.slaName +
            "/sls")).mkdir();
success = (new File(this.getServletContext().getRealPath("/") +
      "ontology/user/" + this.companyName + "/sla/" + this.slaName +
            "/sls/qos")).mkdir();
if (success) {
  System.out.println("criou o subdiretorio sla e sls");
} else {
  System.out.println("Não criou o subdiretorio sla ou  sls ou qos");
}

this.base = URI.create("file:" + this.getServletContext()
      .getRealPath("/") + "ontology/SimpleSLAOnt.owl");
this.manager = OWLManager.createOWLOntologyManager();

//cria a url da ontologia
URI ontologyURI = URI.create(getServletContext().getContextPath() +
      "/ontology/user/" + this.companyName + "/sla/" + this.slaName +
      "/" + this.slaName + "-SLA.owl");

//endereco fisico da ontologia
URI physicalURI = URI.create("file:" + this.getServletContext()
      .getRealPath("/") + "ontology/user/" + this.companyName +
      "/sla/" + this.slaName + "/" + this.slaName + "-SLA.owl");
SimpleURIMapper mapper = new SimpleURIMapper(ontologyURI,
      physicalURI);
this.manager.addURIMapper(mapper);

OWLOntology ontology = this.manager
      .loadOntologyFromPhysicalURI(this.base);
this.walker = new OWLOntologyWalker(Collections.singleton(ontology));
this.visitor = new VisitorSLA(this.walker, ontology, ontologyURI,
      this.companyName, this.borderRouter);
this.walker.walkStructure(this.visitor);

OWLOntology ontAux = ontology;
while (hasMoreImportes(ontAux)) {
  this.importedOntologies = new     ArrayList<OWLOntology>(ontAux
      .getImports(this.manager));
  for (int i = 0; i < this.importedOntologies.size(); i++) {
    ontAux = this.importedOntologies.remove(i);
    this.walker = new
```

```
            OWLOntologyWalker(Collections.singleton(ontAux));
          this.walker.walkStructure(this.visitor);
        }
        this.importedOntologies = new
            ArrayList<OWLOntology>(ontAux.getImports(this.manager));
      }

      this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
            physicalURI);
      this.manager.removeOntology(ontology.getURI());
      request.setAttribute("url", ontologyURI);
      request.setAttribute("companyName", this.companyName);
      request.setAttribute("borderRouter", this.borderRouter);
      request.setAttribute("ip_wan", request.getParameter("ip_wan"));
      request.setAttribute("ip_lan", request.getParameter("ip_lan"));
      request.setAttribute("slaName", this.slaName);
    } catch (OWLOntologyStorageException ex) {
      Logger.getLogger(MakeSLAOntology.class.getName()).log(Level.SEVERE,
      null, ex);
    } catch (UnknownOWLOntologyException ex) {
      Logger.getLogger(MakeSLAOntology.class.getName()).log(Level.SEVERE,
      null, ex);
    } catch (OWLOntologyCreationException ex) {
      Logger.getLogger(MakeSLAOntology.class.getName()).log(Level.SEVERE,
      null, ex);
    }

    RequestDispatcher rd =
      request.getRequestDispatcher("/jsp/view/AddSlsFormView.jsp");
    rd.forward(request, response);

  }

  private boolean hasMoreImportes(OWLOntology ontology) {
    if (ontology.getImports(manager).size() > 0) {
      return true;
    } else {
      return false;
    }
   }
}
```

VISITORSLA.JAVA

```
import java.net.URI;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.model.AddAxiom;
import org.semanticweb.owl.model.OWLAxiom;
import org.semanticweb.owl.model.OWLClass;
import org.semanticweb.owl.model.OWLDataFactory;
import org.semanticweb.owl.model.OWLDataProperty;
import org.semanticweb.owl.model.OWLIndividual;
import org.semanticweb.owl.model.OWLObjectProperty;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyChangeException;
import org.semanticweb.owl.model.OWLOntologyManager;
```

```
import org.semanticweb.owl.util.OWLOntologyWalker;
import org.semanticweb.owl.util.OWLOntologyWalkerVisitor;

/**
 * @author Felipe B. Teixeira
 */
public class VisitorSLA extends OWLOntologyWalkerVisitor<Object> {

    private OWLAxiom ax;
    private AddAxiom addAx;
    private OWLOntology ontology, ontAux;
    private URI ontologyURI;
    private String companyName, borderRouterIP;
    private OWLDataFactory factory;
    private OWLOntologyManager manager;
    private OWLClass CompanySla;
    private OWLIndividual CompanyBorderRouter;
    private List<AddAxiom> changes;

    public VisitorSLA(OWLOntologyWalker walker, OWLOntology ontology, URI
      ontologyURI, String companyName, String borderRouterIP) {
        super(walker);
        this.ontology = ontology;
        this.ontologyURI = ontologyURI;
        this.companyName = companyName;
        this.borderRouterIP = borderRouterIP;
        this.manager = OWLManager.createOWLOntologyManager();
        this.factory = manager.getOWLDataFactory();
        this.CompanySla = factory.getOWLClass(URI.create(this.ontologyURI +
            "#" + this.companyName + "-SLA"));
        this.changes = new ArrayList<AddAxiom>();
    }

    public Object visit(OWLClass desc) {
        if (desc.toString().equals("SLA")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySla,
            desc);
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(this.addAx);
        }
        try {
            this.manager.applyChanges(this.changes);
        } catch (OWLOntologyChangeException ex) {
            Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
            null, ex);
        }
        return null;
    }

    public Object visit(OWLObjectProperty property) {
        try {
            this.manager.applyChanges(this.changes);
        } catch (OWLOntologyChangeException ex) {
            Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
            null, ex);
        }
        return null;
    }

    public Object visit(OWLDataProperty property) {
        if (property.toString().equals("hasProviderName")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySla,
            this.factory.getOWLDataValueRestriction(property,
```

```
            this.factory.getOWLTypedConstant(this.companyName)));
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(this.addAx);
        }

        try {
            this.manager.applyChanges(this.changes);
        } catch (OWLOntologyChangeException ex) {
            Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
            null, ex);
        }
        return null;
    }
}
```

MAKESLSONTOLOGY.JAVA

```
import java.io.File;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Hashtable;
import java.util.List;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.inference.OWLReasonerAdapter;
import org.semanticweb.owl.io.RDFXMLOntologyFormat;
import org.semanticweb.owl.model.AxiomType;
import org.semanticweb.owl.model.OWLAxiom;
import org.semanticweb.owl.model.OWLClass;
import org.semanticweb.owl.model.OWLDescription;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyCreationException;
import org.semanticweb.owl.model.OWLOntologyManager;
import org.semanticweb.owl.model.OWLOntologyStorageException;
import org.semanticweb.owl.model.UnknownOWLOntologyException;
import org.semanticweb.owl.util.CommonBaseURIMapper;
import org.semanticweb.owl.util.OWLOntologyWalker;
import org.semanticweb.owl.util.SimpleURIMapper;
import prototipo.InferenceServer;
import uk.ac.manchester.cs.factplusplus.owlapi.Reasoner;
import uk.ac.manchester.cs.owl.OWLEquivalentClassesAxiomImpl;
import uk.ac.manchester.cs.owl.OWLSubClassAxiomImpl;

/**
 *
 * @author Felipe B. Teixeira
 */
public class MakeSLSOntology extends HttpServlet {
```

```
private String src_addr, src_any, dst_addr, dst_any, form_ontology,
  protocol, other_protocol, src_port, dst_port, dscp_value;
private OWLOntologyManager manager;
private OWLOntologyWalker walker;
private VisitorSLS visitor;
private List<OWLOntology> importedOntologies;
private String slaOntName, companyName,slaName,ip_lan,ip_wan;
private File dir;
private String borderRouter;
private URI slaPhysiclURI, slaOntologyURI, slsPhysicalURI, baseSla;
private static boolean criaSLA = true;
private String providerOntologyURI;

protected void processRequest(HttpServletRequest request,
  HttpServletResponse response) throws ServletException, IOException {
    this.providerOntologyURI =
     "http://www.inf.ufsc.br/~achilles/mestrado/example-provider.owl";
    this.src_addr = request.getParameter("src_addr");
    this.src_any = request.getParameter("src_any");
    this.dst_addr = request.getParameter("dst_addr");
    this.dst_any = request.getParameter("dst_any");
    this.form_ontology = request.getParameter("ontology");
    this.protocol = request.getParameter("protocol");
    this.other_protocol = request.getParameter("other_protocol");
    this.src_port = request.getParameter("src_port");
    this.dst_port = request.getParameter("dst_port");
    this.dscp_value = request.getParameter("dscp_value");
    String[] dados =  request.getParameter("name").split(";");
    this.companyName = dados[0].replaceAll(" ", "_");
    this.slaName = dados[1].replaceAll(" ", "_");
    this.ip_lan = dados[2];
    this.ip_wan = dados[3];

    if (MakeSLSOntology.criaSLA) {
        this.baseSla = URI.create("file:" + this.getServletContext()
              .getRealPath("/") + "ontology/SimpleSLAOnt.owl");
        this.createSLSandSLA();
        MakeSLSOntology.criaSLA = false;
    } else {
        this.baseSla = URI.create("file:" + this.getServletContext()
              .getRealPath("/") + "ontology/SimpleSLAOnt.owl");
        this.createSLS();
        this.baseSla = URI.create("file:" + this
              .getServletContext().getRealPath("/") + "ontology/user/" +
              this.companyName + "/sla/" + this.companyName + "-
              SLA.owl");
        this.alteraSLA();
    }

    this.borderRouter = visitor.getBorderRouter();
    request.setAttribute("url", slaOntologyURI);
    request.setAttribute("companyName", this.companyName);
    request.setAttribute("borderRouter", this.borderRouter);
    request.setAttribute("ip_wan", this.ip_wan);
    request.setAttribute("ip_lan", this.ip_lan);
    request.setAttribute("slaName", this.slaName);
    RequestDispatcher rd =
        request.getRequestDispatcher("/jsp/view/AddSlsFormView.jsp");
    rd.forward(request, response);

}

private boolean hasMoreImportes(OWLOntology ontology) {
```

```
    if (ontology.getImports(manager).size() > 0) {
        return true;
    } else {
        return false;
    }
}

private void createSLSandSLA() {
    try {
        this.manager = OWLManager.createOWLOntologyManager();

        this.baseSla = URI.create("file:" + this.getServletContext()
              .getRealPath("/") + "ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/" + this.slaName + "-SLA.owl");
        //creates the ontologie URL
        URI ontologyURI = URI.create(getServletContext()
              .getContextPath() + "/ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/sls/" + this.companyName + "-
              SLS.owl");
        //Phisical address of the ontology
        String contexto = this.getServletContext().getRealPath("/");
        this.dir = new File(this.getServletContext().getRealPath("/") +
              "ontology/user/" + this.companyName + "/sla/"  +
              this.slaName + "/sls/");
        URI physicalURI = URI.create("file:" + this.getServletContext()
              .getRealPath("/") + "ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/sls/" + this.companyName + "-
              SLS" + this.dir.listFiles().length + ".owl");
        SimpleURIMapper mapper = new SimpleURIMapper(ontologyURI,
              physicalURI);
        this.manager.addURIMapper(mapper);

        this.slaOntologyURI = URI.create(getServletContext()
              .getContextPath() + "/ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/" + this.slaName + "-SLA.owl");
        this.slaPhysiclURI = URI.create("file:" + this
              .getServletContext().getRealPath("/") + "ontology/user/" +
              this.companyName + "/sla/" + this.slaName + "/" +
              this.slaName + "-SLA.owl");
        OWLOntology ontology =
              this.manager.loadOntologyFromPhysicalURI(this.baseSla);

        this.walker = new
              OWLOntologyWalker(Collections.singleton(ontology));
        this.visitor = new VisitorSLS(this.walker, ontology,
              ontologyURI, this.companyName, contexto,this.slaName);

        this.visitor.setDestinationAddress(this.dst_addr);
        this.visitor.setSourceAddress(this.src_addr);
        this.visitor.setDestinationPort(this.dst_port);
        this.visitor.setSourcePort(this.src_port);



        this.walker.walkStructure(this.visitor);
        OWLOntology ontAux = ontology;
        while (hasMoreImportes(ontAux)) {
            this.importedOntologies = new
              ArrayList<OWLOntology>(ontAux.getImports(this.manager));
            for (int i = 0; i < this.importedOntologies.size(); i++) {
                ontAux = this.importedOntologies.remove(i);
                this.walker = new
                    OWLOntologyWalker(Collections.singleton(ontAux));
```

```
                this.walker.walkStructure(this.visitor);
            }
            this.importedOntologies = new
              ArrayList<OWLOntology>(ontAux.getImports(this.manager));
        }
        this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
              physicalURI);
        this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
              this.slaPhysiclURI);
        this.manager.removeOntology(ontology.getURI());

    } catch (OWLOntologyStorageException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
    } catch (UnknownOWLOntologyException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
    } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
    }
}

 private void createSLS() {
    try {
        this.manager = OWLManager.createOWLOntologyManager();
        URI slsOntologyURI =
              URI.create(getServletContext().getContextPath() +
              "/ontology/user/" + this.companyName + "/sla/" +
              this.slaName + "/sls/" + this.companyName + "-SLS.owl");
        //endereco fisico da ontologia
        String contexto = this.getServletContext().getRealPath("/");

        this.dir = new File(contexto + "ontology/user/" +
              this.companyName + "/sla/" + this.slaName + "/sls/");
        this.slsPhysicalURI = URI.create("file:" + this
              .getServletContext().getRealPath("/") + "ontology/user/" +
              this.companyName + "/sla/" + this.slaName+ "/sls/" +
              this.companyName + "-SLS" + this.dir.listFiles().length +
              ".owl");
        SimpleURIMapper mapper = new SimpleURIMapper(slsOntologyURI,
              slsPhysicalURI);
        this.manager.addURIMapper(mapper);
        OWLOntology ontology =
              this.manager.loadOntologyFromPhysicalURI(this.baseSla);

        this.walker = new
              OWLOntologyWalker(Collections.singleton(ontology));
        this.visitor = new VisitorSLS(this.walker, ontology,
              slsOntologyURI, this.companyName, contexto,this.slaName);
        //Seta todas os valores para os parametros das sls
        this.visitor.setDestinationAddress(this.dst_addr);
        this.visitor.setSourceAddress(this.src_addr);
        this.visitor.setDestinationPort(this.dst_port);
        this.visitor.setSourcePort(this.src_port);
        this.walker.walkStructure(this.visitor);
        OWLOntology ontAux = ontology;
        while (hasMoreImportes(ontAux)) {
            this.importedOntologies = new
              ArrayList<OWLOntology>(ontAux.getImports(this.manager));
            for (int i = 0; i < this.importedOntologies.size(); i++) {
                ontAux = this.importedOntologies.remove(i);
                this.walker = new
```

```
                    OWLOntologyWalker(Collections.singleton(ontAux));
                this.walker.walkStructure(this.visitor);
            }
            this.importedOntologies = new
              ArrayList<OWLOntology>(ontAux.getImports(this.manager));
        }
        this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
              slsPhysicalURI);
        this.manager.removeOntology(ontology.getURI());

    } catch (OWLOntologyStorageException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
        System.out.println("Problems saving the ontology");
    } catch (UnknownOWLOntologyException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
    } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
        System.out.println("Problem creating the ontology");
    }
}

private void alteraSLA() {
    try {
        this.manager = OWLManager.createOWLOntologyManager();
        URI slsOntologyURI = URI
              .create(getServletContext().getContextPath() +
              "/ontology/user/" + this.companyName + "/sla/" +
              this.slaName + "/sls/" + this.companyName + "-SLS.owl");
        //endereco fisico da ontologia
        String contexto = this.getServletContext().getRealPath("/");
        this.slaOntologyURI = URI.create(getServletContext()
              .getContextPath() + "/ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/" + this.slaName + "-SLA.owl");
              this.slaPhysiclURI = URI.create("file:" + contexto +
              "ontology/user/" + this.companyName + "/sla/" +
              this.slaName + "/" + this.slaName + "-SLA.owl");
        SimpleURIMapper mapper = new
              SimpleURIMapper(this.slaOntologyURI, this.slaPhysiclURI);
        this.manager.addURIMapper(mapper);

        OWLOntology ontology =
              this.manager.loadOntologyFromPhysicalURI(this.baseSla);
        this.walker = new
              OWLOntologyWalker(Collections.singleton(ontology));
        this.visitor = new VisitorSLS(this.walker, ontology,
              slsOntologyURI, this.companyName, contexto,this.slaName);
        //Seta todas os valores para os parametros das sls
        this.visitor.setDestinationAddress(this.dst_addr);
        this.visitor.setSourceAddress(this.src_addr);
        this.visitor.setDestinationPort(this.dst_port);
        this.visitor.setSourcePort(this.src_port);
        this.walker.walkStructure(this.visitor);
        OWLOntology ontAux = ontology;
        while (hasMoreImportes(ontAux)) {
            this.importedOntologies = new
              ArrayList<OWLOntology>(ontAux.getImports(this.manager));
            for (int i = 0; i < this.importedOntologies.size(); i++) {
                ontAux = this.importedOntologies.remove(i);
                this.walker = new
                    OWLOntologyWalker(Collections.singleton(ontAux));
```

```
                this.walker.walkStructure(this.visitor);
            }
            this.importedOntologies = new
        ArrayList<OWLOntology>(ontAux.getImports(this.manager));
        }

        this.manager.saveOntology(ontology, new RDFXMLOntologyFormat(),
              this.slaPhysiclURI);
        this.manager.removeOntology(ontology.getURI());

    } catch (OWLOntologyStorageException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
        System.out.println("Problems saving the ontology");
    } catch (UnknownOWLOntologyException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
    } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(MakeSLSOntology.class.getName()).log(Level.SEVE
              RE, null, ex);
        System.out.println("Problem creating the ontology");
    }
 }

protected void doPost(HttpServletRequest request, HttpServletResponse
   response)
        throws ServletException, IOException {
     processRequest(request, response);
 }

 public String getServletInfo() {
     return "Short description";
 }

 private Hashtable<OWLClass, ArrayList<OWLClass>>
   matchingProviderSpecifications(String clientOntologyURIStr, String
   providerOntologyURIStr) {
   System.load("/usr/lib/jvm/java-6-sun-
        1.6.0.16/jre/lib/i386/libFaCTPlusPlusJNI.so");
   //Instantiate Ontology manager
   this.manager = OWLManager.createOWLOntologyManager();
   String netQoSOntBaseURIStr =
        "http://www.inf.ufsc.br/~achilles/mestrado/";
   String ontologiesLocalDir = this.getServletContext().getRealPath("/")
        + "ontology/ontologiesPrototipo/";
   System.out.println(ontologiesLocalDir);
   CommonBaseURIMapper mapper = new
        CommonBaseURIMapper(URI.create("file://" + ontologiesLocalDir));
   OWLReasonerAdapter reasoner = null;
   File dirOnt = new File(ontologiesLocalDir);
   String localOntologyFiles[] = dirOnt.list();
   for (String file : localOntologyFiles) {
        mapper.addMapping(URI.create(netQoSOntBaseURIStr +
        file.toString()), ontologiesLocalDir + file.toString());
}

this.manager.addURIMapper(mapper);

try {
    //Instantiate Reasoner
    reasoner = new Reasoner(this.manager);
} catch (Exception ex) {
    Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
```

```
          null, ex);
    System.out.println("Error instatiating the reasoner");
}
//That logging is really annoying
Logger l = Logger
    .getLogger("uk.ac.manchester.cs.factplusplus.owlapi.OntologyLoader");
l.setLevel(Level.OFF);

Hashtable<OWLClass, ArrayList<OWLClass>> matchingProviderSpecs = new H
    ashtable<OWLClass, ArrayList<OWLClass>>(); //Assuring the return value
try {
    OWLClass qosSpec = this.manager.getOWLDataFactory()
          .getOWLClass(URI.create(netQoSOntBaseURIStr +
          "BaseOnt.owl#QoSSpec"));
    OWLOntology clientOntology = this.manager.loadOntologyFromPhysicalURI
          (URI.create(clientOntologyURIStr));
    ArrayList<Object> clientSpecs = new ArrayList<Object>();

    for (OWLAxiom axiom : clientOntology.getAxioms()) {
        if (axiom.getReferencedEntities().contains(qosSpec)) {
            if (axiom.getAxiomType().equals(AxiomType.EQUIVALENT_CLASSES))
            {
                OWLEquivalentClassesAxiomImpl aux =
                (OWLEquivalentClassesAxiomImpl) axiom;
                clientSpecs.addAll(aux.getNamedClasses());
            } else if (axiom.getAxiomType().equals(AxiomType.SUBCLASS)) {
                OWLSubClassAxiomImpl aux = (OWLSubClassAxiomImpl)
                axiom;
                clientSpecs.add(aux.getSubClass());
            }
        }
    }
    Set<OWLOntology> clientOntologyImportsClosure =
          this.handleOntology(clientOntology);
    Set<OWLOntology> providerOntologyImportsClosure =
          this.handleOntologyString(providerOntologyURIStr);

    reasoner.loadOntologies(providerOntologyImportsClosure);
    reasoner.loadOntologies(clientOntologyImportsClosure);

    System.out.println("Classifying ontologies (and measuring Reasoner
          response time)");
    double start = System.currentTimeMillis();
    reasoner.classify();
    double elapsedTime = System.currentTimeMillis() - start;
    System.out.println("done.");
    System.out.println("Cassified in" + elapsedTime + "ms");

    OWLClass owlNothing = this.manager.getOWLDataFactory()
          .getOWLClass(URI.create("http://www.w3.org/2002/07/owl#Nothing")
          );


    ArrayList providerSpecs;
    for (Object clientSpec : clientSpecs) {
        providerSpecs = new ArrayList();
        providerSpecs.addAll(OWLReasonerAdapter.flattenSetOfSets(reasoner.
          getDescendantClasses((OWLDescription) clientSpec)));
        providerSpecs.addAll(reasoner.getEquivalentClasses((OWLDescription
          ) clientSpec));
        providerSpecs.removeAll(clientSpecs);
        providerSpecs.remove(owlNothing);
        matchingProviderSpecs.put((OWLClass) clientSpec, providerSpecs);
```

```
        }

        this.dir = new File(this.getServletContext().getRealPath("/") +
        "ontology/user/" + this.companyName + "/sla/" + this.slaName +
        "/sls/qos/");
        URI clientOntologyPhysicalURI = URI.create("file:" +
              this.getServletContext().getRealPath("/") + "ontology/user/" +
              this.companyName + "/sla/" + this.slaName + "/sls/qos/" +
              this.companyName + "-SLS" + (this.dir.listFiles().length + 1) +
              ".owl");
        this.slaPhysiclURI = URI.create("file:" + this.getServletContext()
              .getRealPath("/") + "ontology/user/" + this.companyName +
              "/sla/" + this.slaName + "/" + this.slaName + "-SLA.owl");
        if(matchingProviderSpecs.size() > 0)
            this.manager.saveOntology(clientOntology, new
              RDFXMLOntologyFormat(), clientOntologyPhysicalURI);


    } catch (OWLOntologyCreationException ex) {
        Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
              null, ex);
    } catch (Exception ex) {
        Logger.getLogger(InferenceServer.class.getName()).log(Level.SEVERE,
              null, ex);
    }
    return matchingProviderSpecs;
}

        private Set<OWLOntology> handleOntology(OWLOntology ontology) throws
              Exception {
          String netQoSOntBaseURIStr =
              "http://www.inf.ufsc.br/~achilles/mestrado/";
          Set<OWLOntology> ontologyImportsClosure =
              this.manager.getImportsClosure(ontology);
          if (!ontologyImportsClosure
              .contains(this.manager.getOntology(URI.create(netQoSOntBaseURISt
              r +"BaseOnt.owl")))) {
              throw new Exception("Not NetQoSOntDerivedException " +
                    ontology.getURI().toString());
          }
        return ontologyImportsClosure;
        }

        private Set<OWLOntology> handleOntologyString(String ontologyString) {
              OWLOntology ontology = null;
          Set<OWLOntology> ontologyImportsClosure = null;
          try {
              ontology = this.manager.loadOntologyFromPhysicalURI
                    (URI.create(ontologyString));
              ontologyImportsClosure = this.handleOntology(ontology);
          } catch (OWLOntologyCreationException ex) {
              Logger.getLogger(InferenceServer.class.getName())
                    .log(Level.SEVERE,null, ex);
          } catch (Exception ex) {
              Logger.getLogger(InferenceServer.class.getName())
                    .log(Level.SEVERE, null, ex);
        }
        return ontologyImportsClosure;
        }
}
```

VISITORSLS.JAVA

```
import sla.VisitorSLA;

import java.io.File;
import java.net.URI;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.semanticweb.owl.apibinding.OWLManager;
import org.semanticweb.owl.model.AddAxiom;
import org.semanticweb.owl.model.OWLAxiom;
import org.semanticweb.owl.model.OWLClass;
import org.semanticweb.owl.model.OWLConstant;
import org.semanticweb.owl.model.OWLDataFactory;
import org.semanticweb.owl.model.OWLDataProperty;
import org.semanticweb.owl.model.OWLDataPropertyExpression;
import org.semanticweb.owl.model.OWLIndividual;
import org.semanticweb.owl.model.OWLObjectProperty;
import org.semanticweb.owl.model.OWLOntology;
import org.semanticweb.owl.model.OWLOntologyChangeException;
import org.semanticweb.owl.model.OWLOntologyManager;
import org.semanticweb.owl.util.OWLOntologyWalker;
import org.semanticweb.owl.util.OWLOntologyWalkerVisitor;

/**
 * @author Felipe B. Teixeira
 */
public class VisitorSLS extends OWLOntologyWalkerVisitor<Object> {

    private OWLAxiom ax;
    private AddAxiom addAx;
    private OWLOntology ontology;
    private URI ontologyURI;
    private OWLDataFactory factory;
    private OWLOntologyManager manager;
    private List<AddAxiom> changes;
    private OWLClass CompanySLS, CompanySLA, CompanySpecification;
    private String companyName, borderRouter,slaName;
    private File dir;
    private boolean firstTime;
    private OWLIndividual FlowID,Scope,DestinationPort
      ,SourcePort,DestinationIPAddress,SourceIPAddress;
    private String dst_addres_value, src_address_value;

    public VisitorSLS(OWLOntologyWalker walker, OWLOntology ontology, URI
      ontologyURI, String companyName,String contexto ,String slaName ) {
        super(walker);
        this.ontology = ontology;
        this.ontologyURI = ontologyURI;
        this.companyName = companyName;
        this.manager = OWLManager.createOWLOntologyManager();
        this.factory = manager.getOWLDataFactory();
        this.firstTime = true;
        this.slaName = slaName;
        this.dir = new File(contexto   + "ontology/user/" + this.companyName
```

```
        +"/sla/" + this.slaName + "/sls/");
    this.CompanySLS = factory.getOWLClass(URI.create(this.ontologyURI +
        "#" + this.companyName + "-SLS" + dir.listFiles().length));
    this.dir = new File(contexto   + "ontology/user/" + this.companyName
        +"/sla/" + this.slaName + "/sls/");//por enquanto eh sls, ateh
        ter specification (qos)
    this.CompanySpecification =            factory.getOWLClass(URI
        .create(this.ontologyURI + "#" + this.companyName + "-
        Specification" +  this.dir.listFiles().length ));
    this.changes = new ArrayList<AddAxiom>();
    this.FlowID = factory.getOWLIndividual(URI.create(this.ontologyURI +
        "#" + this.companyName + "-FlowID"));
    this.Scope = factory.getOWLIndividual(URI.create(this.ontologyURI +
        "#" + this.companyName + "-Scope"));

    List<OWLClass> aux = new ArrayList<OWLClass>(this
        .ontology.getReferencedClasses());
    for (int i = 0; i < aux.size(); i++) {
        if (aux.get(i).toString().equals(this.companyName + "-SLA")) {
            this.CompanySLA = aux.get(i);
            break;
        }
    }
    this.CompanySLA = factory.getOWLClass(URI.create(this.ontologyURI +
        "#" + this.companyName + "-SLA"));
}

public Object visit(OWLClass desc) {
    if (desc.toString().equals("SLS")) {
        this.ax = this.factory.getOWLSubClassAxiom(this.CompanySLS,
             desc);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(this.addAx);
    }
    if (desc.toString().equals("QoSSpec") && this.firstTime) {
        this.firstTime = false;
        this.ax = this.factory
        .getOWLSubClassAxiom(this.CompanySpecification, desc);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(this.addAx);
    }
    if (desc.toString().equals("Scope")) {
        this.ax = this.factory.getOWLClassAssertionAxiom(this.Scope,
        desc);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(this.addAx);
    }
    if (desc.toString().equals("FlowID")) {
        this.ax = this.factory.getOWLClassAssertionAxiom(this.FlowID,
        desc);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(this.addAx);
    }

    if (desc.toString().equals("DestinationIPAddress")) {
        this.ax = this.factory
        .getOWLClassAssertionAxiom(this.DestinationIPAddress, desc);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(this.addAx);
    }

    if (desc.toString().equals("SourceIPAddress")) {
        this.ax = this
```

```
                .factory.getOWLClassAssertionAxiom(this.SourceIPAddress,
                desc);
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(this.addAx);
        }
        try {
            this.manager.applyChanges(this.changes);
        } catch (OWLOntologyChangeException ex) {
            Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
            null, ex);
        }
        return null;
    }

    public Object visit(OWLObjectProperty property) {
        if (property.toString().equals("hasSLS")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySLA, this
            .factory.getOWLObjectSomeRestriction(property,
            this.CompanySLS));
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(this.addAx);
        }

        if (property.toString().equals("hasSpecification")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySLS, this
                  .factory.getOWLObjectSomeRestriction(property,
                  this.CompanySpecification));
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(this.addAx);
        }

        if (property.toString().equals("hasFlowID")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySLS,
            this.factory.getOWLObjectValueRestriction(property,
            this.FlowID));
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(addAx);
        }

        if (property.toString().equals("hasScope")) {
            this.ax = this.factory.getOWLSubClassAxiom(this.CompanySLS, this
                  .factory.getOWLObjectValueRestriction(property,
                  this.Scope));
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(addAx);
        }

        if(property.toString().equals("hasDestinationIPAddress")){
            this.ax = this
                  .factory.getOWLObjectPropertyAssertionAxiom(this.Scope,
                  property, this.DestinationIPAddress);
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(addAx);
        }

        if(property.toString().equals("hasSourceIPAddress")){
            this.ax = this
                  .factory.getOWLObjectPropertyAssertionAxiom(this.Scope,
                  property, this.SourceIPAddress);
            this.addAx = new AddAxiom(this.ontology, this.ax);
            this.changes.add(addAx);
        }
```

```
    try {
        this.manager.applyChanges(this.changes);
    } catch (OWLOntologyChangeException ex) {
        Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
        null, ex);
    }
    return null;
}

public Object visit(OWLDataProperty property){
    if (property.toString().equals("hasIdentifier")) {
        this.ax = this
              .factory.getOWLDataPropertyAssertionAxiom(this.SourceIPAdd
              ress, property,this.src_address_value);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(addAx);
        this.ax = this
              .factory.getOWLDataPropertyAssertionAxiom(this.Destination
              IPAddress, property,this.dst_addres_value);
        this.addAx = new AddAxiom(this.ontology, this.ax);
        this.changes.add(addAx);
    }

    try {
        this.manager.applyChanges(this.changes);
    } catch (OWLOntologyChangeException ex) {
        Logger.getLogger(VisitorSLA.class.getName()).log(Level.SEVERE,
        null, ex);
    }
    return null;
}

public Object visit(OWLIndividual individual) {
    if (individual.toString().equals(this.companyName + "-
        BorderRouter")) {
        Map mapAux = new
        TreeMap(individual.getDataPropertyValues(ontology));
        Iterator keys = mapAux.keySet().iterator();
        OWLDataPropertyExpression property;
        while (keys.hasNext()) {
            property = (OWLDataPropertyExpression) keys.next();
            if (property.toString().equals("hasIdentifier")) {
                List<OWLConstant> setAux = new
                ArrayList<OWLConstant>((Set<OWLConstant>)
                    mapAux.get(property));
                this.borderRouter = setAux.get(0).getLiteral();
                break;
            }
        }
    }
    return null;
}

public String getBorderRouter() {
    return this.borderRouter;
}

public void setDestinationPort(String dst_port){
    this.DestinationPort =
        factory.getOWLIndividual(URI.create(this.ontologyURI +
        "#dst_port:" + dst_port));
}
```

```
    public void setSourcePort(String src_port){
        this.SourcePort =
            factory.getOWLIndividual(URI.create(this.ontologyURI +
            "#src_port:" + src_port));
    }

    public void setDestinationAddress(String dst_address){
        this.DestinationIPAddress =
            factory.getOWLIndividual(URI.create(this.ontologyURI +
            "#dst_address:" + dst_address));
        this.dst_addres_value = dst_address;

    }

    public void setSourceAddress(String src_address){
        this.SourceIPAddress =
            factory.getOWLIndividual(URI.create(this.ontologyURI +
            "#src_address:" + src_address));
        this.src_address_value = src_address;
    }

}
```

ANEXO B - ARTIGO

Negociação de Acordos de Nível de Serviços em Redes de Computadores

Felipe Bottan Teixeira[1]

[1]Laboratório de Pesquisa em Sistemas Distribuídos – LaPeSD
Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina – UFSC
Florianópolis – SC - Brasil

{felipecomp19}@inf.ufsc.br

Abstract. *Communication services with quality of service (QoS) are often a essential requisite for the users of this type of service. And when there is guarantee of QoS, the client and the network service providers must negotiate a Service Level Agreement (SLA). Nowadays this negotiation is done based on specific quality parameters of the provider. This work consist in an implementation of a prototype of a web application for the negotiation of service level specification offering a greater flexibility in terms of parameters to specify a QoS during the negotiation of a SIP (Session Initiation Protocol) session. This application is meant to be a prototype for QoS systems, which makes use of a semantic approach to the formal specification of the quality parameters.*

Resumo. *Serviços de comunicação com garantias de qualidade de serviço (QoS) são cada vez mais um requisito essencial para os usuários deste tipo de serviço. E quando há garantia de QoS, os clientes e os provedores de serviço de rede devem negociar um Acordo de Nível de serviço (SLA – Service Level Agreement). Atualmente esta negociação e feita com base em parâmetros de qualidade específicos do provedor. Este trabalho consiste na implementação de um protótipo de negociação de acordos de Nível de serviço para uma aplicação web, oferecendo uma maior flexibilidade em termos de parâmetros para especificação da QoS durante a negociação de uma sessão SIP (do inglês Session Initiation Protocol). Este protótipo, baseado em avanços na área de Web Semântica, fará uso de ontologias para representar a especificação dos parâmetros de qualidade e tem o objetivo de ser um protótipo formal para sistemas de QoS.*

1. Introdução

Um dos principais requisitos dos serviços de Voz sobre IP (VoIP), bem como em muitos outros serviços multimídia, é a garantia de Qualidade de Serviço (QoS). O serviço de comunicação deve garantir limites em termos de diversos parâmetros de QoS (p.e., vazão, atraso, taxa de perda de pacotes) para que a qualidade da voz seja aceitável. Estes limites são definidos em um Acordo de Nível de Serviço (SLA – Service Level Agreement) firmado entre o cliente e a Provedora de Serviço de Rede (NSP – Network Service Provider). Um SLA contém uma lista de Especificação do Nível de Serviço (SLS - Service Level Agreement) que especificam um conjunto de parâmetros de qualidade, seus valores e suas métricas que variam de acordo com o tipo de Serviço prestado.

Atualmente não existe nenhuma normalização para a Especificação de QoS, assim como não existe uma lista de parâmetros de qualidade adotadas universalmente para compor uma SLS. Algumas soluções adotam uma lista fixa de parâmetros, porém uma devido a heterogeneidade da aplicações e diferentes arquiteturas adotados pelos provedores, soluções com uma lista de extensível de parâmetros tem se mostrado mais eficiente. A princípio, no processo de negociação do serviço, o cliente e a aplicação devem ser capazes de especificar a QoS utilizando-se de parâmetros quantitativos e qualitativos. Parâmetros esses que devem ser compreendidos e mapeados de forma a serem reconhecidos pelos NSPs.

Esse mapeamento consegue-se através do uso de ontologias. Que pode ser definida como um conjunto de conceitos, relações, instâncias e axiomas, usado para descrever um domínio de interesse e permite a descrição formal da informação, o mapeamento entre terminologia distintas e também a interpretação automática de parâmetros de QoS.

O objetivo do presente trabalho é oferecer um mecanismo para compor SLAs dinâmicos, oferecendo maior flexibilidade no momento de escolha da QoS e, a cima de tudo, um serviço com qualidade. Para isso será utilizada uma abordagem semântica na criação da SLA, utilizando-se a NetQoSOnt, ontologia desenvolvida por Prudêncio (2010) e tem como meta principal permitir a especificação da QoS com transparência em termos de parâmetros e oferece suporte a interpretação automática dos parâmetros de QoS através do uso de inferência em ontologias.

As seções que seguem estão organizadas da seguinte maneira. Nas seções 2 e 3 é feita uma revisão bibliográfica sobre Qualidade de Serviço e sobre a ontologia NetQoSOnt. Posteriormente, na seção 4 será apresentado um modelo para negociação de SLA/SLS utilizando-se de uma abordagem semântica e finalmente, na seção 5, serão apresentadas as conclusões obtidas.

2. Qualidade de Serviço

Atualmente, em uma arquitetura TCP/IP básica, exite apenas uma classe de serviço, denominada Melhor Esforço (do inglês *Best Effort),* onde não existe nenhuma garantia em termos de parâmetros de rede. Visando suprir essa deficiência foram propostas, pela IETF (*Internet Engineering Task Force*), novas arquiteturas de gerenciamento de QoS onde seja possível oferecer garantias em termos de parâmetros de rede.

Segundo Royer (2008) as principais arquiteturas de gerenciamento de QoS propostas pela IETF foram:

MPLS (*MultiProtocol Label Switching)*

Fornece uma melhoria no tráfego da rede através de uma melhoria na função de roteamento, reduzindo o overhead e as latências nos roteadores, porém não existe nenhuma garante em termos dos parâmetros de rede mencionados.

IntServ (*Integrated Services)*

No modelo IntServ (Integrated Services) Braden (1994), estabelece a implantação de uma infraestrutura para a Internet com o intuito de suportar o transporte de áudio, vídeo e dados em tempo real além do tráfego de dados atual. Nesta arquitetura a QoS e garantida através de mecanismos de reserva de recurso na rede, onde a aplicação deve reservar os recursos que deseja utilizar antes de iniciar o envio dos dados, usando o protocolo RSVP (Resourse Reservation Protocol).

O protocolo de sinalização RSVP atua sobre o tráfego de pacotes em uma rede TCP/IP (Internet, redes privadas, entre outras). É considerado um protocolo eficiente em relação a

garantia de QoS a medida que provê granularidade e controle fino das solicitações feitas pelas aplicações, tendo como maior desvantagem a sua complexidade de operação nos roteadores que, eventualmente, podem gerar problemas nos backbones de redes maiores, como das Operadoras (SILVA, 2005).

DiffServ (*Differentiated Services)*

A arquitetura DiffServ (Differentiated Services) (BLAKE, 1998) foi introduzida pelo IETF devido as limitações na arquitetura IntServ. No DiffServ os pacotes são marcados diferentemente, na entrada da rede, para criar classes de pacotes. As classes recebem serviços, ou tratamentos diferenciados na rede, conforme definições preestabelecidas. Serviços Diferenciados é essencialmente um esquema de priorização de pacotes.

Nesta arquitetura, a diferenciação dos serviços de rede é feita através da especificação e marcação do campo DS do cabeçalho do data grama IP. Os Serviços Diferenciados são oferecidos no interior de um domínio DS, que é composto por um conjunto de nós compatíveis com a proposta DiffServ. No DiffServ, a marcação do campo DS indica o tipo de serviço que deverá ser provido pelos sistema de comunicação para o pacote.

A especificação deste serviço pode ser feito com base na SLA/SLS e segundo BARBIARZ, CHAN e BAKER (2006) o conjunto de classes de serviço (CoS) definidas para o um domínio DiffServ são *Default Forwarding*, que pode garantir uma vazão mínima, mas sem ordem ou entrega dos pacotes, *Assured Forwarding*, que é sugerida para aplicações que necessitam de confiabilidade e prioridade maior no tráfego de dados, e a de maior prioridade de todas *Expedited Forwarding*, que fornece um serviço com baixo atraso, baixa variação de atraso, baixa perda de pacotes e vazão mínima.

3. NetQoSOnt

Em diversas operações relacionadas ao gerenciamento da QoS, é necessário um meio eficiente de especificar QoS. A grande diversidade de soluções de QoS utilizadas pelas NSPs, cada uma com terminologia e definições próprias, torna difícil o desenvolvimento de soluções de negociação de QoS válida em todos os cenários.

Este trabalho fez uso da ontologia NetQoSOnt apresentada por Prudêncio (2010) que se propõe a ser uma ontologia para especificação de qualidade no domínio de redes de computadores, possibilitando a criação de novas especificações e também a interpretação e comparação destas de forma automatizada.

Esta ontologia foi desenvolvida utilizando a OWL 2.0, nova versão proposta pela W3C, que supera diversas limitações das versões anteriores de OWL utilizadas pelas ontologias de QoS. E como ocorre com outras ontologias, esta continua sendo atualizada devido a necessidade de especificação de novos recursos que por ventura se fizerem necessários em novas aplicações.

A NetQoSOnt é uma ontologia de base a ser utilizada no gerenciamento de QoS oferecendo uma maior flexibilidade em termos de parâmetros de qualidade. Esta ontologia propõe classes e módulos que permitem expressar vários conceitos relacionados a especificação de QoS. Seu desenvolvimento foi influenciado pela organização em camadas dos protocolos de rede, o que permitiu uma redução da complexidade na implantação destes protocolos e na diferenciação dos serviços de rede.

4. Negociação de SLA/SLS baseado em semântica

Uma SLA tem o seu ciclo de vida divido em três fases. Fase de criação, onde o cliente e a NSP

estão cientes do serviço contratado. Fase operacional, que é o momento de utilização do serviço. E nesse momento para que o cliente possa definir a qualidade de serviço desejada, ou seja, especificar a SLS, é necessário uma interface aonde ele possa alterar a SLS(s) que compõem a SLA. E por fim a Fase de Remoção, onde, após o término do serviço, todas as configuração necessária para a garantia de qualidade devem ser desfeitas.

Para o controle do ciclo de vida e da validação da SLA é necessário um sistema de negociação de SLA/SLS denominado *Bandith Broker (*ou Gerenciador de Recursos). Este sistema é composto por quatro módulos. O Módulo de Negociação de SLA/SLS , que é a interface com o cliente para a criação da SLA. O Módulo de Inferência responsável pela inferência das ontologias de SLA criadas pelo módulo de negociação. O Módulo AAA(do inglês *Authentication*, *Authorization* e *Accouting)* responsável pelo monitoramento dos recursos de rede a fim de verificar se o ambiente suporta a classe de serviço que foi inferida e também o Módulo Gerenciador NETCONF responsável pela realização das configurações necessárias para atender a qualidade desejada.

4.1. Protótipo de Negociação de SLA/SLS

Para validação do sistema proposto foi desenvolvido um protótipo onde foram implementados dois dos quatro módulos citados na secção anterior. Neste protótipo foram implementados os módulos de negociação e de inferência, e por não influenciar no resultado da solução proposta foram desconsiderados os termos burocráticos de uma SLA, como por exemplo a informações das partes, valor do serviço, etc.

Este protótipo foi desenvolvido utilizando a linguagem JAVA, a ontologia NetQoSOnt como base para a especificação da QoS, a OWL API, uma API para manipulação de ontologias dentro das normas da W3C e também foi utilizado o *reasoner* (ou motor de inferência) Pellet.

Neste protótipo o cliente pode criar uma SLA (veja figura 1) dando um identificador para ela e informando o IP WAN e o IP LAN desta SLA. O IP WAN seria o IP do roteador de borda da rede, que é o roteador responsável pela classificação dos pacotes no domínio DiffServ, e o IP LAN seria o IP interno da máquina onde a SLA está sendo criada. Neste mesma interface o cliente pode inserir as ontologias de especificação de qualidade de serviço (link *Add QoS ontology)* e também as de identificação de fluxo (link *Add Flow Identification ontology)* que irão compor a base de conhecimento desta SLA e serão as SLSs que posteriormente poderão ser selecionadas para compor essa SLA.

Uma vez criada a SLA o cliente pode então compor este contrato com suas especificações de nível de serviço ou SLSs. O cliente pode optar por uma SLS estática, ,que poderia ser uma SLS padrão ou uma previamente utilizada, ou pode optar por uma SLS dinâmica (veja figura 2).

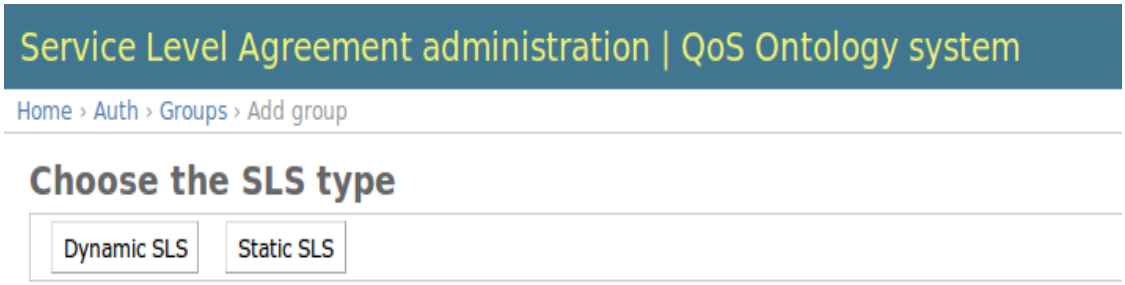

Figura 2 - Interface para definição do tipo da SLS

Uma vez selecionado o tipo dinâmico o cliente é direcionado para uma nova página (veja figura 3) aonde ele poderá definir o escopo desta SLS (*Source Address* e *Destination Address),* optar pela qualidade de serviço desejada (*QoS Specification)* e identificar o fluxo (*Flow Identification).* Nos dois últimos campos serão listadas somente as ontologias adicionadas anteriormente a base de conhecimento na interface de criação da SLA.

Uma vez definidos os campos o cliente pode salvar essa ontologias, e neste momento será realizada um inferência da ontologia do cliente com a do provedor afim de verificar se o provedor de serviço atende a qualidade e o tipo de serviço especificado pelo cliente. E exista uma classe de serviço deste NSP que atenda a qualidade será feito uma *cache* da QoS inferida e inicia-se o serviço.

4.2 Testes

Para validação do protótipo foram realizados teste em um cenário composto por dois roteadores Diffserv, dois cliente VoIP, um provedor VoIP Asterix, e um Sistema de Gerência de QoS onde foram utilizados os dois módulos propostos por este trabalho e foram simulados os módulos AAA e o Gerenciador NETCONF.

Durante os testes foram realizadas 10 chamadas VOIP, sendo 5 delas sem garantia de qualidade de serviço e 5 dela com garantias. A classe inferida foi a EF (*Expedited Forwarding),* a de maior prioridade, e para captura de dados para analise foi utilizado o software WireShark.

Os testes mostraram que leva-se 14ms para a realização da solicitação da chamada sem a garantia de QoS e 578ms com a garantia de qualidade. Porém, caso haja uma nova chamada com a mesma especificação de qualidade, esse tempo será reduzido para 446 ms pois é realizado um *cache* da classe inferida. Sendo assim não é necessário uma nova inferência da ontologia. E notou-se que leva-se 20 ms para, após a solicitação da chamada, a qualidade de serviço esteja sendo realmente entregue.

5. Conclusões

Neste trabalho foi proposto um modelo para negociação de acordos de nível de serviço onde, durante o período de validade do contrato (ou SLA) o usuário e o provedor do serviço possam criar, modificar e remover as especificações de serviço (SLS), oferecendo um maior dinamismo na hora de definição da qualidade de serviço e dando ao usuário um maior poder de escolha do nível de QoS desejado.

A comparação de parâmetros de QoS é realizada automaticamente por um motor de inferência. Ao carregar a ontologia que contém o pedido do cliente e a ontologia que contém as CoSs do provedor, o cálculo da nova hierarquização das classes permite ao provedor saber quais de suas CoSs atendem ao pedido do cliente simplesmente consultando o motor de

inferência por quais de suas CoSs que foram inferidas como subclasses ou classes equivalentes da classe que representa especificação do cliente.

A utilização de uma abordagem semântica para especificação de qualidade de serviço mostrou-se eficiente e permite a flexibilização no momento de escolha da QoS. E os testes mostraram que o tempo gasto, praticamente 0,5 segundos, para oferecer um serviço com qualidade, é um tempo aceitável.

Referências

D. Black et al. An Architecture for Differentiated Services, RFC 2475, Internet Engineering Task Force, 1998.

Royer, J., Willrich, R. e Diaz, M. (2008). "User Profile-Based Authorization Policies for Network QoS Services". 7th IEEE Int. Symp. on Network Computing and Applications (NCA), pp. 68-75.

K. Chan, et al. COPS Usage for Policy Provisioning (COPS-PR), RFC 3084, 2001.

R. Enns. NETCONF Configuration Protocol, RFC 4741, Internet Engineering Task Force, 2006.

G. Camarillo, W. Marshall, J. Rosenberg. Integration of Resource Management and Session Initiation Protocol (SIP), RFC 3312, Internet Engineering Task Force, 2002.

J. Polk, S. Dhesikan, G. Camarillo. Quality of Service (QoS) Mechanism Selection in the Session Description Protocol (SDP), RFC 5432, Internet Engineering Task Force, 2009.

H.J. Park, J.J. Yang, J.K. Choi, H.S. Kim. QoS Negotiation for IPTV Service using SIP. 9th Int. Conf. on Advanced Communication Technology, pp. 945-948, 2007.

M. Alexander, P. Suppan. An Architecture for SDP-based Bandwidth Resouce Allocation with QoS for SIP in IEEE 802.16 Networks. 2nd Int. Workshop on Quality of Service & Sercurity for Wireless and Mobile Networks, pp. 75-82, 2006.

J. Rosenberg et al. SIP: Session Initiation Protocol, RFC 3261, Internet Engineering Task Force, 2002.

M. Handley et al. SDP: Session Description Protocol, RFC 4566, Internet Engineering Task Force, 2006.

R. Willrich, L.H. Vicente, R.B. Uriarte, A.C. Prudêncio, J. Cé Júnior. Invocação Dinâmica de Serviços com QoS em Sessões Multimídia SIP. 8th Int. Information and Telecommunication Technologies Symposium (I2TS), 2009.

A.C. Prudêncio, R. Willrich, S. Tazi , M. Diaz. Quality of Service Specifications: A Semantic Approach. In: 8th IEEE International Symposium on Network Computing and Application, pp. 219-226, 2009.

P. Calhoun, et al. Diameter Base Protocol, RFC 3588, Internet Engineering Task Force, 2003.

D. Sun et al. Diameter Quality of Service Application, RFC 5866, Internet Engineering Task Force, 2010.

The Open Source PBX & Telephony Platform. URL: http://www.asterisk.org/, 2010.

SIP Inspector. http://sourceforge.net/projects/sipinspector/, 2010.

X-Lite. http://www.counterpath.com/x-lite.html, 2010.

Wireshark. http://www.wireshark.org/, 2010.