Final

IP Network Usage Accounting - Parte 2
Accounting system

PROJETO DE MESTRADO

Luís Miguel Correia Ferreira
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

outubro | 2014

IP Network Usage Accounting - Parte 2
Accounting system

PROJETO DE MESTRADO

Luís Miguel Correia Ferreira
MESTRADO EM ENGENHARIA INFORMÁTICA

SUPERVISOR
Lina Maria Pestana Leão de Brito
Filipe Fernandes Azevedo

CO-SUPERVISOR
Karolina Baras

Resumo

O controlo de banda larga é um conceito importante quando lidamos com redes de larga escala. Os ISPs precisam de garantir disponibilidade e qualidade de serviço a todos os clientes, enquanto garantem que a rede como um todo não fica mais lenta.

Para garantir isto, é necessário que os ISPs recolham dados de tráfego, analisem-nos e usem-nos para definir a velocidade de banda larga de cada cliente.

A NOS Madeira implementou, durante vários anos, um sistema semelhante. No entanto, este sistema encontrava-se obsoleto, sendo necessário construir um novo, totalmente de raíz. Entre as limitações encontrava-se a impossibilidade de alterar os algoritmos de análise de tráfego, fraca integração com os serviços de gestão de rede da NOS Madeira e reduzida escalabilidade e modularidade. O sistema *IP Network Usage Accounting* é a resposta a estes problemas.

Este projeto foca-se no desenvolvimento do subsistema *Accounting System*, o segundo dos três subsistemas que compõem o sistema *IP Network Usage Accounting*. Este subsistema, implementado com sucesso e atualmente em produção na NOS Madeira, é responsável por analisar os dados referidos acima e usar os resultados dessa análise para direcionar a disponibilidade de banda larga, de acordo com o uso da rede de cada cliente.

Palavras-chave

Controlo de largura de banda

Bases de dados

QoS

Tráfego web

Abstract

Bandwidth management is an important concept when dealing with large networks. ISP's need to guarantee availability and quality of service to every client, all while ensuring the network as a whole doesn't slow down.

To ensure this, it is necessary for ISP's to have the ability to collect traffic data, analyze it and use it to shape the broadband speed of each individual client.

NOS Madeira implemented such a system, for several years. However, this system became deprecated, and with it arised the need for a new system. Amongst the limitations was the impossibility to alter the traffic analysis algorithms, weak integration with NOS Madeira's network management systems and reduced scalability and modularity. The IP Network Usage Accounting system is the answer to these problems.

This project focuses on the development of the Accounting System, the second of the three subsystems which make up the IP Network Usage Accounting system. This subsystem, which was successfully implemented and is currently in production in NOS Madeira, is responsible for analyzing the data mentioned above, and use the results of this analysis to steer bandwidth availability, according to each client's usage of the network.

Keywords

Bandwidth Management

Databases

QoS

Network Traffic

Agradecimentos

Gostaria de agradecer às minhas orientadoras, a Professora Lina Maria Pestana Leão de Brito e a Professora Karolina Baras, por todo o encorajamento que me deram, por terem sempre procurado dar o seu melhor para me ajudar e, principalmente, por terem acreditado em mim.

Gostaria de agradecer a todos os Professores que me deram aulas ao longo destes seis anos de curso na Universidade da Madeira, transmitindo os vossos valiosos conhecimentos e puxando por mim sempre até ao fim.

Gostaria de agradecer aos meus co-orientadores na NOS Madeira, Nélio Vieira e Filipe Azevedo, por toda a ajuda e motivação que me deram, e também pelos conhecimentos transmitidos.

Gostaria de agradecer aos meus colegas Daniel Aguiar e Jorge Canha, que colaboraram neste projecto, pela disponibilidade e altruísmo de cada vez que colocavam o seu trabalho de lado para ajudar no meu.

Gostaria de agradecer aos meus pais, irmã e namorada, por terem paciência e compreensão, de cada vez que não lhes pude dispensar o tempo necessário por ter que trabalhar no projeto.

Mas acima de tudo, quero agradecer a todos vós, e ao resto da minha familia e amigos aqui não mencionados. Sem vocês, sem o contributo que cada um deixou na minha vida, não seria possível chegar onde cheguei. A todos vós, estou eternamente grato.

Índice

Lista de Figuras

Lista de *Snippets*

Lista de acrónimos

API – *Application Programming Interface*;

CPE – *Customer Premises Equipment*;

CRUD – *Create, Read, Update, Delete*;

CSV – *Comma-Separated Values*;

DPI – *Deep Packet Inspection*;

DTS – *Data Transformation Services*;

FTTH – *Fiber To The Home*;

HFC – *Hybrid-Fibre Coaxial*;

HTML – *HyperText Markup Language*;

HTTP – *HyperText Transfer Protocol*;

IP – *Internet Protocol*;

ISP – *Internet Service Provider*;

JSON – *JavaScript Object Notation*;

LAMP – *Linux, Apache, MySQL, PHP*;

MAC - *Media Access Control*;

MVC – *Model-View-Controller*;

OLE DB – *Object Linking and Embedding, Database*;

OSS – *Operations Support System*;

QoS – *Quality of Service*;

RSS - *Really Simple Syndication*;

SCE – *Service Control Engine*;

SMTP – *Simple Mail Transfer Protocol*;

SOAP – *Simple Object Access Protocol*;

SQL – *Structured Query Language*;

URL – *Uniform Resource Locator*;

W3C – *World Wide Web Consortium*;

WSDL – *Web Services Description Language*;

WWW – *World Wide Web*;

XML – e*Xtensible Markup Language*;

Capítulo 1 - Introdução

A regulação do consumo de banda larga por parte dos ISPs (*Internet Service Providers*) é um desafio difícil que pode levar a frustrações por parte dos clientes. A linha que separa o fornecimento de um serviço adequado e eficaz da utilização abusiva é muito ténue, sendo necessário aplicar extremo cuidado na disponibilização de serviços mais rápidos ou mais lentos, respetivamente.

A NOS Madeira é o ISP líder na Madeira, fornecendo Internet de banda larga através de redes HFC (*Hybrid Fibre-Coaxial*), FTTH (*Fiber To The Home*) e 3G. De forma a manter estas redes a funcionar sem problemas e garantindo uma experiência de primeira classe aos utilizadores, estatísticas de uso da rede eram geradas continuamente e processadas por algoritmos específicos que, por sua vez, interagiam com a própria rede, alocando a disponibilidade da largura de banda para onde esta era mais necessária. Estas ações eram feitas automaticamente por uma plataforma que estava dividida logicamente em três componentes, nomeadamente o componente coletor de estatísticas de uso da rede, o componente de análise de dados e o componente de controlo da rede. Estes componentes corriam independentemente como três subsistemas, interagindo entre eles através de APIs (*Application Programming Interfaces*) bem definidas.

Embora o sistema anterior realizasse as tarefas mencionadas, este encontrava-se tecnologicamente obsoleto, tinha falhas, como a falta de interfaces para integração com um OSS (*Operations Support System*), não permitia a implementação de algoritmos diferentes para recolher e processar estatísticas de tráfego, e não era escalável, sustentável ou modular como deveria ser, para responder a todas as necessidades da NOS Madeira.

O novo sistema *IP Network Usage Accounting* está dividido em três subsistemas, que interagem entre si e com elementos exteriores para recolher os dados de tráfego, manipulá-los e controlar o QoS (*Quality of Service*) dos clientes. Estes subsistemas estão idealizados de forma a poderem ser distribuídos por máquinas distintas, ou efetuarem o seu processamento na mesma máquina. Devido à complexidade de desenvolvimento dos mesmos, estes subsistemas foram desenvolvidos por três pessoas diferentes.

O primeiro subsistema, desenvolvido por Daniel Aguiar, é o *IP Mapping*, responsável pela recolha dos dados de tráfego e descrito em [1].

O segundo subsistema, em foco neste projeto, é o *Accounting System*, e tem como responsabilidade principal efetuar cálculos com o tráfego dos clientes para chegar a um valor de tráfego ponderado, atribuindo um QoS consoante o resultado.

O terceiro subsistema, desenvolvido por Jorge Canha, é o *Policy Server*, descrito em [2] responsável por receber do *Accounting System* o mapeamento de QoS por IP (*Internet Protocol*), e aplicar o mesmo aos clientes da NOS Madeira.

1.1 - Objetivos

O objetivo principal deste projeto é definir os requisitos para o novo sistema de análise de dados de tráfego, com o envolvimento dos diversos *stakeholders*, desenhar a arquitetura do subsistema e implementá-lo de acordo com os padrões de engenharia de *software* atuais, usando teoria de sistemas distribuídos e padrões de desenho de software comuns, como o MVC (*Model-View-Controller*).

É necessário permitir que, para cada produto disponibilizado aos clientes, possam ser definidas diferentes qualidades de serviço conforme o tráfego efetuado, e implementados novos algoritmos para contabilizar esse tráfego, em vez de apenas um conjunto predefinido para todos os produtos, como era usado no sistema anterior. Estes novos algoritmos deverão ser criados recorrendo a variáveis que podem ser definidas arbitrariamente. Por fim, pretende-se a disponibilização de serviços web que possam responder às várias necessidades dos operadores e da própria criação e definição de regras a aplicar aos produtos.

1.2 - Estrutura do documento

Este documento encontra-se dividido em seis capítulos, organizados da seguinte forma:

No Capítulo 1 será feita uma breve contextualização, descrição do sistema a desenvolver, um resumo dos 3 subsistemas e quais os objetivos do projeto.

No Capítulo 2 será feita uma análise ao estado da arte, descrevendo o sistema anterior e as tecnologias que serviram de suporte à conceção do novo sistema.

No Capítulo 3 será abordada a modelação e uma descrição pormenorizada do *Accounting System.* Também será feita a análise de requisitos.

No Capítulo 4 será abordada a implementação do sistema, descrevendo as ferramentas usadas, detalhes da base de dados, opções/decisões tomadas ao longo do processo de desenvolvimento, quais as soluções encontradas e também código relevante para complementar as explicações.

No Capítulo 5 serão apresentados os resultados da implementação, os problemas mais relevantes que surgiram, os testes efetuados para garantir a qualidade do sistema e alguns dados estatísticos.

No Capítulo 6 serão descritas as conclusões e como poderá o sistema ser melhorado no futuro.

Capítulo 2 – Tecnologias relacionadas

Neste capítulo será abordado o sistema usado anteriormente para a contagem de tráfego. De seguida, haverá uma descrição das tecnologias que servem de suporte ao projeto. Embora estas tecnologias tenham sido definidas de início pela NOS Madeira, sem possibilidade de escolha, também serão abordadas alternativas possíveis.

Foi também feito um estudo sobre sistemas semelhantes. Embora seja do conhecimento geral que estes existem e encontram-se implementados em diversos ISPs, a informação encontrada foi escassa e com pouco detalhe no que toca a procedimentos internos, como os cálculos e os componentes dos sistemas. Por esta razão, o presente documento não faz referência a sistemas semelhantes.

2.1 - Sistema anterior

A regulação de largura de banda (*bandwidth throttling* em inglês) é o ato de diminuir ou aumentar a largura de banda intencionalmente de um serviço de Internet por um ISP, para gerir e controlar a utilização de banda larga. Este controlo visa evitar abusos na utilização excessiva da rede, disponibilizando mais largura de banda a utilizadores que façam menos tráfego, e menos largura de banda aos que a congestionam. O sistema de contabilidade de tráfego implementado anteriormente tinha diversas limitações, que deram lugar à necessidade de implementar um sistema completamente novo de raiz. Essas limitações ocorriam sobretudo devido ao uso de tecnologias já obsoletas e pouco flexíveis, o que impedia uma simples atualização dos componentes do sistema e implicou que este fosse todo construído de raiz. As principais limitações deste sistema serão descritas de seguida.

2.1.1 - Principais limitações do sistema existente

O sistema anterior corria em Windows 2000 e Microsoft SQL Server 2000. Ambos os sistemas estão descontinuados e já não têm qualquer suporte por parte da Microsoft [3], o que pode levar a falhas de segurança que não serão corrigidas. Além disso, devido à idade e descontinuação dos produtos, estes sistemas obviamente não implementam as funcionalidades necessárias atualmente.

O sistema anterior estava completamente escrito em VBScript e DTS (*Data Transformation Services*). VBScript é uma linguagem em desuso, sendo utilizada maioritariamente em sistemas legados. A partir do Internet Explorer 11, a Microsoft recomenda que qualquer página que use VBScript seja atualizada para usar *JavaScript*, caso contrário a página poderá não ser exibida corretamente [4].

OS DTS fornecem funcionalidade para importar, exportar e transformar dados entre um servidor SQL e uma API OLE DB (*Object Linking and Embedding, Database*) [5]. OLE DB é uma API usada para aceder a dados que podem estar em vários formatos, desde bases de dados, a folhas de cálculo e até mesmo ficheiros de texto (como exemplo, Microsoft Access e Microsoft

Excel) [6]. Os DTS estão obsoletos, tendo a Microsoft os substituído por *SQL Server Integration Services* [7]. Um exemplo de uma interface para DTS pode ser visto na Figura 1:

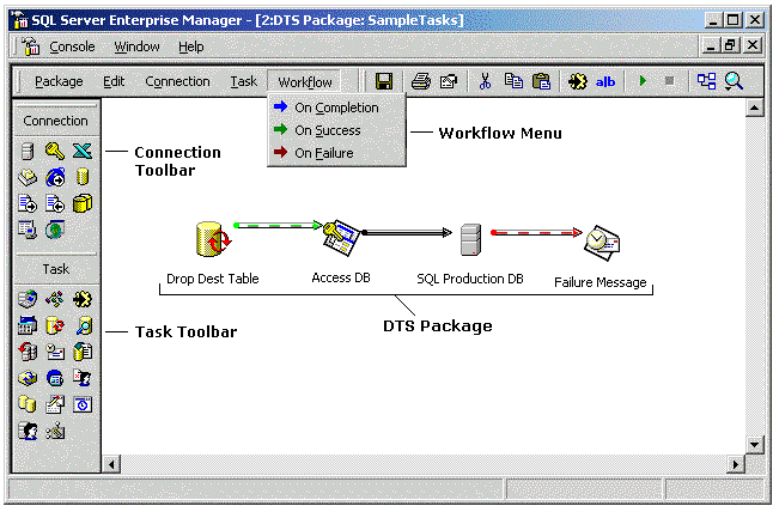

Figura 1 - Exemplo de interface para DTS

Além dos pontos referidos anteriormente, o sistema anterior também tinha diversas limitações a nível das operações que podia fazer e como podia gerir os dados que recebia. Essas limitações eram:

- 7 níveis fixos de prioridade – Ou seja, todos os produtos ofereciam os mesmos tipos de QoS, sem qualquer diferença, sendo 7 o nível mais alto e com maior largura de banda, e 1 o nível mais baixo e com menor largura de banda. As regras para calcular os níveis podiam ser diferentes entre produtos, mas no fim o nível 7 do produto x dava a mesma largura de banda que o nível 7 do produto y;
- Fronteiras entre níveis únicas para todos os produtos – As fronteiras entre níveis representam o nível máximo de tráfego (em *Bytes*, embora aqui seja exemplificado em *Gigabytes*) que um determinado cliente pode efetuar para pertencer a um determinado nível. Por exemplo, para estar no nível 7 só pode fazer no máximo 2GB de tráfego, para estar no nível 6 só pode fazer no máximo 4GB de tráfego e para estar no nível 5 só pode fazer no máximo 8GB de tráfego. Como as fronteiras entre níveis eram iguais para todos os produtos, sem hipótese de alterá-las, era impossível definir que num determinado produto a prioridade só baixa do nível 7 para o 6 quando este faz 6GB de tráfego em vez de 2GB, por exemplo;
- Impossibilidade de alterar as regras de negócio (que serão descritas no Capítulo 3);
- Mesmas regras de negócio para todos os produtos – Embora os parâmetros recebidos nas regras de negócio variassem entre cada produto, a lógica das regras em si era a mesma entre todos. Por exemplo, todos os produtos implementavam a regra de negócio que "corta" o tráfego efetuado às 2h da manhã numa determinada percentagem. A regra é sempre a mesma, mas a percentagem varia de produto para produto;
- Não guardava dados de tráfego para fins legais – Existem ocasiões em que a NOS recebe pedidos de informação sobre um determinado IP/cliente. Com o sistema antigo, era outra plataforma que estava encarregue desta função;

- Não podia responder ao SCE (*Service Control Engine*) quando este perguntava por um IP desconhecido – Esta situação levava a inconsistências no QoS atribuído ao IP em questão, pois o SCE só sabia que QoS atribuir a um IP após receber essa informação do sistema. Sendo impossível responder ao pedido, o SCE tinha que atribuir um QoS por defeito;

De seguida, serão abordadas as tecnologias usadas no novo sistema, além de alternativas às mesmas.

2.2 - Serviços Web

Um serviço web permite que uma aplicação possa fazer pedidos a outra aplicação, até mesmo quando estão ambas em plataformas diferentes e são programadas em linguagens diferentes. Isto é conseguido recorrendo a protocolos como o SOAP (*Simple Object Access Protocol*) [8]. Estes serviços são de extrema importância neste projeto, pois é através dos mesmos que os três subsistemas conseguem comunicar (independentemente de estarem na mesma máquina ou não) e fornecer informações ao OSS, como por exemplo, no caso do *Accounting System*, informação sobre quanto tráfego um determinado cliente fez nas últimas 24h, ou a variação dos seus níveis de prioridade nos últimos 7 dias.

2.2.1 - SOAP

Segundo a W3C (*World Wide Web Consortium*), organização responsável pelos padrões da WWW (*World Wide Web*), o protocolo SOAP "é usado para trocar informações em ambientes descentralizados e distribuídos" [9]. É um protocolo baseado em XML (e*Xtensible Markup Language*) e está dividido em 3 partes:

- Um envelope que define uma *framework* para descrever os conteúdos das mensagens e como processá-las;
- Um conjunto de regras de codificação para tipos de dados;
- Uma convenção para representar chamadas de procedimento remotas;

O envelope SOAP contém a função a chamar no lado do servidor. Por exemplo, o pedido pode invocar uma função que retorna qual o QoS atual de um determinado *MAC Address* (*Media Access Control Address*), enviando-o como argumento no pedido. O protocolo SOAP pode ser usado com HTTP (*Hypertext Transfer Protocol*) ou até mesmo SMTP (*Simple Mail Transfer Protocol*). Embora seja complexo construir o envelope de pedido manualmente, várias ferramentas já possuem funcionalidade para gerar automaticamente os pedidos, tomando como *input* os parâmetros necessários após análise ao ficheiro WSDL (*Web Services Description Language*) do serviço web. Embora o SOAP tenha tratamento de erros incorporado, esta automatização do processo é bastante útil [10].

Simplificando, uma mensagem SOAP é um documento XML que é usado para transmitir dados ou invocar operações remotamente. No *Snippet* 1 podemos ver um exemplo de um pedido SOAP:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soap="http://stnetaccounting.nosmadeira.com/businessrules/index/soap">
   <soapenv:Header/>
   <soapenv:Body>
      <soap:currentPackage soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <macAddress xsi:type="xsd:string">00265b1d00e0</macAddress>
      </soap:currentPackage>
   </soapenv:Body>
</soapenv:Envelope>
```

Snippet 1 - Exemplo de um envelope de um pedido SOAP

Como podemos verificar, ao fazer o pedido, indicamos qual a função ou serviço que queremos invocar, enviando um ou mais argumentos. O envelope com o pedido SOAP é transmitido através da rede (por exemplo, através de HTTP) e lido pelo servidor, que retorna outro envelope SOAP com a resposta ao pedido.

Esta resposta contém uma estrutura de dados, composta com o nome e tipo dos campos, além do valor. No exemplo apresentado no *Snippet* 2, a resposta é composta por números inteiros e *strings*. Cabe ao cliente interpretar o ficheiro XML corretamente e usar os dados obtidos para as suas necessidades.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns1="http://stnetaccounting.nosmadeira.com/businessrules/index/soap"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ns2="http://xml.apache.org/xml-soap"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
      <ns1:currentPackageResponse>
         <return xsi:type="SOAP-ENC:Struct">
            <errorId xsi:type="xsd:int">0</errorId>
            <errorText xsi:type="xsd:string">OK</errorText>
            <isOverride xsi:type="xsd:int">0</isOverride>
            <currentPackage xsi:type="ns2:Map">
               <item>
                  <key xsi:type="xsd:string">subnet</key>
                  <value xsi:type="xsd:string">83.223.189.30/32</value>
               </item>
               <item>
                  <key xsi:type="xsd:string">sceQos</key>
                  <value xsi:type="xsd:string">29</value>
               </item>
               <item>
                  <key xsi:type="xsd:string">internalQos</key>
                  <value xsi:nil="true"/>
               </item>
               <item>
                  <key xsi:type="xsd:string">accountedTraffic</key>
                  <value xsi:type="xsd:string">0</value>
               </item>
            </currentPackage>
         </return>
      </ns1:currentPackageResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Snippet 2 - Exemplo de um envelope de uma resposta SOAP

2.2.2 - WSDL

A WSDL é uma linguagem em XML usada para descrever um serviço web. Esta indica quais as mensagens e operações usadas pelos serviços, quais os parâmetros que são enviados/recebidos, se o serviço envia uma resposta e qual o tipo da resposta que é suposto obter, e ainda a forma como é feita a ligação ao serviço [11]. É importante para o funcionamento do protocolo SOAP, pois é a partir dos ficheiros WSDL que os clientes podem ver detalhes sobre os serviços web fornecidos, além de que é através do WSDL que as ferramentas conseguem gerar envelopes de pedidos SOAP bem formados. No *Snippet* 3 podemos ver um exemplo de um ficheiro WSDL:

```
<portType name="Businessrules_Model_ServicesPort">
    <operation name="currentPackage">
        <documentation>Returns the current package of a given MAC Address</documentation>
        <input message="tns:currentPackageIn"/>
        <output message="tns:currentPackageOut"/>
    </operation>
</portType>
<binding name="Businessrules_Model_ServicesBinding" type="tns:Businessrules_Model_ServicesPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="currentPackage">
        <soap:operation soapAction="http://stnetaccounting.nosmadeira.com/businessrules/index/soap#currentPackage"/>
        <input>
            <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://stnetaccounting.nosmadeira.com/businessrules/index/soap"/>
        </input>
        <output>
            <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://stnetaccounting.nosmadeira.com/businessrules/index/soap"/>
        </output>
    </operation>
</binding>
<service name="Businessrules_Model_ServicesService">
    <port name="Businessrules_Model_ServicesPort" binding="tns:Businessrules_Model_ServicesBinding">
        <soap:address location="http://stnetaccounting.nosmadeira.com/businessrules/index/soap"/>
    </port>
</service>
<message name="currentPackageIn">
    <part name="macAddress" type="xsd:string"/>
</message>
<message name="currentPackageOut">
    <part name="return" type="xsd:struct"/>
</message>
```

Snippet 3 - Exemplo de ficheiro WSDL

2.2.3 - REST

O REST (*Representational State Transfer*) *é* um estilo arquitetural muitas vezes aplicado a serviços web, recorrendo a HTTP. Qualquer API de serviços web que use REST é intitulada de "RESTful". Para testar uma API REST, muitas vezes basta um URL (bem formado) que indique qual a operação a ser executada e os parâmetros necessários. O REST também pode fazer uso dos comandos HTTP GET, POST, PUT e DELETE para efetuar as operações [10]. Por outras palavras, basta aceder via *browser* para testar a API do serviço [12], como exemplificado na Figura 2:

Figura 2 - Exemplo de um serviço web RESTful

Aqui a função "*currentPackage*" faz parte do URL (*Uniform Resource Locator*) e o *MAC Address* está na parte seguinte do mesmo. Ao aceder a este URL, a página que é retornada contém as informações pedidas. O resultado pode vir em CSV (*Comma-Separated Values*), JSON (*JavaScript Object Notation*) e RSS (*Really Simple Syndication*) [10]. No *Snippet* 4 podemos ver um exemplo de uma resposta ao pedido em formato JSON:

```
{
    "errorId":"0",
    "errorText":"OK",
    "isOverride":"0",
    "subnet":"83.223.189.30/32",
    "sceQos":"29",
    "internalQos":"true",
    "accountedTraffic":"0"
}
```

Snippet 4 - Resposta em JSON a serviço web RESTful

2.2.4 – SOAP VS REST

Ao escolher qual o protocolo a usar, é útil ter em conta as vantagens de cada um. O protocolo SOAP tem as seguintes vantagens [10] [13]:

- Independente de linguagem, plataforma e transporte (enquanto o REST precisa de usar HTTP);
- Todas as mensagens obedecem a um padrão/protocolo de envio e receção;
- Tratamento de erros incorporado;
- Certas ferramentas ajudam a automatizar o processo de criação dos envelopes SOAP;
- Implementa mais funcionalidades de segurança;
- Suporta ACID (*Atomicity, Consistency, Isolation, Durability*), enquanto REST não;
- Implementa o protocolo *WS-ReliableMessaging*, que permite que mensagens SOAP sejam entregues sem problemas entre aplicações distribuídas, mesmo que hajam erros no *software*, sistema ou rede. Por sua vez, o REST não lida com esta situação, obrigando o utilizador a efetuar o pedido novamente, o que pode levar a maus resultados se o primeiro pedido tiver sido executado pelo serviço (embora sem retornar resposta);

Por sua vez, o protocolo REST tem vantagem nos seguintes pontos [10] [13]:

- Fácil de aprender;
- Mensagens mais curtas para a mesma informação, e consequentemente com menos tamanho (SOAP tem que usar XML enquanto o REST pode usar formatos de mensagem mais pequenos);
- Rápido a executar, sem necessidade de muito processamento;
- Mais flexível, não tendo que obedecer tão rigidamente a padrões pré-definidos;
- Os resultados de um pedido REST ficam em *cache*, enquanto os de um pedido SOAP não;

A escolha entre SOAP e REST depende das necessidades de cada situação. No entanto, no âmbito do projeto, e visto que é necessário mais que alto desempenho, uma alta fiabilidade, a

solução mais correta é o protocolo SOAP, por este ter tratamento de erros incorporado, suportar ACID e ter um sistema de entrega de mensagens mais fiável que o REST.

2.3 – Sistema de gestão de base de dados relacional

Existem vários sistemas gestores de bases de dados relacionais. MySQL, PostgreSQL, SQLite, MariaDB, para nomear alguns. No entanto, nesta secção apenas será abordado o MySQL, por ser o sistema usado na NOS Madeira.

O MySQL é um dos sistemas de gestão de bases de dados relacionais de código aberto mais usados atualmente. É bastante popular em aplicações web, sendo utilizado por empresas como a Google [14] e Facebook [15]. Nas secções seguintes serão abordados dois temas importantes no contexto de bases de dados: as propriedades ACID e os dois motores de base de dados mais populares em MySQL.

2.3.1 - Propriedades ACID

ACID, ou *Atomicity*, *Consistency*, *Isolation* e *Durability* (atomicidade, consistência, isolação e durabilidade) é um acrónimo usado para definir um conjunto de propriedades que garantem que transações em bases de dados são efetuadas corretamente. Andreas Reuter e Theo Härder, que criaram o acrónimo em 1983 (embora o conceito já existisse previamente) descrevem as propriedades da maneira que a seguir se descreve:

2.3.1.1 - *Atomicity*

Uma transação tem que ser "tudo ou nada", ou seja, uma transação ou é completamente efetuada ou nada é alterado. O utilizador deve saber, seja qual for o resultado, em que estado está [16].

2.3.1.2 - *Consistency*

Uma transação que chegue ao seu fim de forma correta, fazendo COMMIT aos seus resultados, preserva a consistência da base de dados. Por outras palavras, cada transação bem-sucedida faz COMMIT apenas a resultados válidos [16].

2.3.1.3 - *Isolation*

Eventos numa transação devem estar escondidos de outras transações que corram paralelamente. Caso contrário uma transação não poderia voltar ao seu estado inicial. Isto é conseguido através de sincronização [16].

2.3.1.4 - *Durability*

A partir do momento em que uma transação está completa e é feito COMMIT à base de dados, o sistema deve garantir que estes resultados sobrevivem falhas subsequentes. Ou seja, o utilizador deve ter uma garantia que as operações que o sistema diz ter feito aconteceram realmente e os resultados ficaram armazenados permanentemente [16].

Na secção seguinte serão abordados os dois motores de dados mais populares em MySQL: o InnoDB e o MyISAM.

2.3.2 - Motor de base de dados

A escolha do motor de base de dados é sempre de extrema importância. O motor é o software que permite que um sistema de base de dados faça operações CRUD, *Create, Read, Update, Delete* (criar, ler, atualizar, apagar). Durante bastante tempo, mais especificamente até à versão 5.4 do MySQL, o motor que vinha configurado por defeito e também o mais usado (a grande maioria dos utilizadores não o alterava) era o MyISAM [17]. A partir do MySQL 5.5, o motor por defeito passou a ser o InnoDB e, no caso do subsistema *Accounting*, a escolha foi facilitada por um conjunto de requisitos que só o mesmo podia satisfazer. Visto que estes dois motores são os mais usados, é feita aqui uma comparação para ver as vantagens e desvantagens de cada um:

- O motor MyISAM bloqueia completamente uma tabela onde esteja a ser feito um UPDATE ou DELETE em qualquer registo [18]. O InnoDB apenas bloqueia o próprio registo [19]. Rapidamente se percebe que, para um sistema que tem que suportar várias operações concorrentes, o motor MyISAM é uma escolha errada, que só atrasará o processamento;
- O motor InnoDB recupera de falhas através dos seus logs: quaisquer alterações que tivessem feito *COMMIT* antes da falha são finalizadas, enquanto operações que estavam a ser processadas na altura da falha são desfeitas. O MyISAM tem que fazer uma verificação completa e reparar ou reconstruir quaisquer índices ou mesmo tabelas que foram atualizadas, mas cujos conteúdos não foram completamente guardados em disco. À medida que as bases de dados ficam maiores, este processo demora cada vez mais no MyISAM, enquanto no InnoDB o tempo de recuperação é fixo, tornando-o a melhor escolha para um sistema que armazena uma grande quantidade de dados, como o subsistema *Accounting* [17];
- O motor InnoDB organiza as tabelas por chaves primárias. As operações em que a coluna da chave primária (ou colunas para chaves conjuntas) seja referenciada são automaticamente otimizadas e o desempenho é bastante alto [17];
- O motor InnoDB suporta transações [19], ao contrário do MyISAM [18]. Se o subsistema *Accounting* usasse o último, seria impossível garantir a integridade dos dados devido às diversas operações que têm que ser executadas de forma atómica, principalmente nos cálculos. De forma resumida, o MyISAM não suporta operações ACID enquanto o InnoDB suporta;
- Em InnoDB, se os dados em disco ficarem corrompidos, um mecanismo de *checksum* lança um alerta antes de se usarem os dados afetados [17];

- O InnoDB *buffer pool* coloca em *cache* dados de índices e tabelas à medida que estes são acedidos. Os dados que são usados frequentemente são processados diretamente a partir da memória. Esta *cache* é usada em vários tipos de informação e otimiza/acelera consideravelmente o processamento, ao ponto de servidores de base de dados dedicados atribuírem 80% da sua memória física ao InnoDB *buffer pool* [17];
- O InnoDB suporta chaves estrangeiras, que garantem integridade referencial entre diferentes tabelas. Isto impede que sejam inseridos dados numa tabela secundária sem haver dados correspondentes na primeira [17];

2.4 – Conclusão

Este capítulo abordou o sistema anterior, permitindo uma maior perceção da importância de uma nova solução, ao documentar as suas características e limitações. Desta forma, é também realçado o ponto em que é necessário um sistema construído completamente de raiz.

De seguida, foram abordadas tecnologias de serviços web que representam possíveis soluções para a implementação do projeto, descrevendo as suas vantagens e desvantagens para ser possível efetuar uma comparação correta.

Finalmente, foi abordado o sistema de gestão de bases de dados relacionais MySQL, apresentando algumas características que o tornam um forte candidato a solução de base de dados no projeto.

Segue-se o Capítulo 3, em que será apresentada a modelação do sistema e o estudo feito antes de começar a sua implementação.

Capítulo 3 – Desenho e Modelação

Previamente à programação do sistema, foram efetuadas diversas reuniões para compreender e acertar detalhes acerca do funcionamento do mesmo. Foram criados diagramas de atividades e de sequência. No entanto, com o decorrer da implementação do projeto, muitas das ideias iniciais foram substituídas por outras mais adequadas à realidade do problema. Os diagramas originais podem ser consultados nos Anexos.

Segundo Sommerville [20], a modelação de um sistema "é o processo de desenvolver modelos abstratos do sistema, com cada modelo a representar uma vista ou perspetiva diferente. (…) O aspeto mais importante do modelo de um sistema é que deixa de fora os detalhes. Idealmente, a representação de um sistema deverá manter toda a informação sobre a entidade a ser representada. Uma abstração simplifica-a deliberadamente e escolhe as características mais salientes."

É de notar que a modelação feita pode não obedecer totalmente às regras de cada representação, o que não é um erro mas sim um ato deliberado para adaptar os modelos às necessidades do problema. Sommerville [20] defende esta abordagem ao referir que "quando se desenvolvem modelos de sistema, podemos ser flexíveis na forma que a notação gráfica é usada. Não é necessário restringir-se aos detalhes da notação. O detalhe e rigor de um modelo depende de como o queremos usar."

De seguida, apresenta-se a arquitectura geral do sistema *IP Network Usage Accounting* e a modelação do subsistema a desenvolver no âmbito deste projecto.

3.1 - Arquitetura geral do sistema

O sistema *IP Network Usage Accounting*, composto por três subsistemas, tem como objetivo analisar o tráfego dos clientes da NOS Madeira, atribuir um QoS de acordo com regras definidas para o produto de cada cliente e garantir que estes cumprem a política de utilização aceitável do ISP.

O primeiro subsistema é o *IP Mapping*, responsável pela recolha dos dados de tráfego e efetuar a associação entre IPs e *MAC Addresses*.

O segundo subsistema tem como responsabilidade principal efetuar cálculos com o tráfego dos clientes para chegar a um valor de tráfego ponderado, e atribuir um QoS consoante o resultado.

O terceiro subsistema é o *Policy Server*, responsável por receber o mapeamento de QoS por IP do *Accounting System* e aplicar o mesmo aos clientes da rede, através do SCE da Cisco.

Os três subsistemas do sistema *IP Network Usage Accounting* comunicam com o OSS, fornecendo serviços web de consulta e manipulação de dados via SOAP. Estas relações estão exemplificadas na Figura 3, que mostra o diagrama componente-conector do sistema.

Para uma melhor compreensão da estrutura de rede que serve de suporte ao projeto, serão descritos sucintamente os seus componentes. Estes componentes são os mesmos tanto no sistema anterior, como no sistema desenvolvido neste projeto:

- OSS – Um OSS (*Operations Support System*) é um conjunto de programas para ajudar um ISP a monitorizar, controlar, analisar e gerir uma rede telefónica ou de computadores [21];
- *Netflow Server* - É uma tecnologia que envia os dados de fluxo de tráfego para um servidor para registo e/ou tratamento. Consiste em IP de origem, porta de origem, IP de destino, porta de destino e tamanho do *flow* em *bytes;*
- *Subscriber Manager* – O Subscriber Manager (SM) é um componente de middleware proprietário da Cisco que fornece informações (IP e package) dos subscribers a uma ou mais plataformas SCE (*Service Control Engine*) [2];
- Cisco SCE – O Cisco SCE é um DPI (*Deep Packet Inspector*). É um dispositivo que permite a classificação, análise e controlo do tráfego feito por IP [22];

O *Netflow Server* recolhe dados sobre as comunicações, com os IPs origem e destino, e associa os IPs dos assinantes da rede aos seus respetivos *MAC Addresses*. Estes dados são recolhidos pelo subsistema *IP Mapping*, que os envia para o *Accounting System*. Após efetuar os cálculos e atribuir um QoS a cada *MAC Address*, o *Accounting System* envia os dados para o *Policy Server*, que comunica com o SCE para atribuir o QoS correto aos clientes da rede.

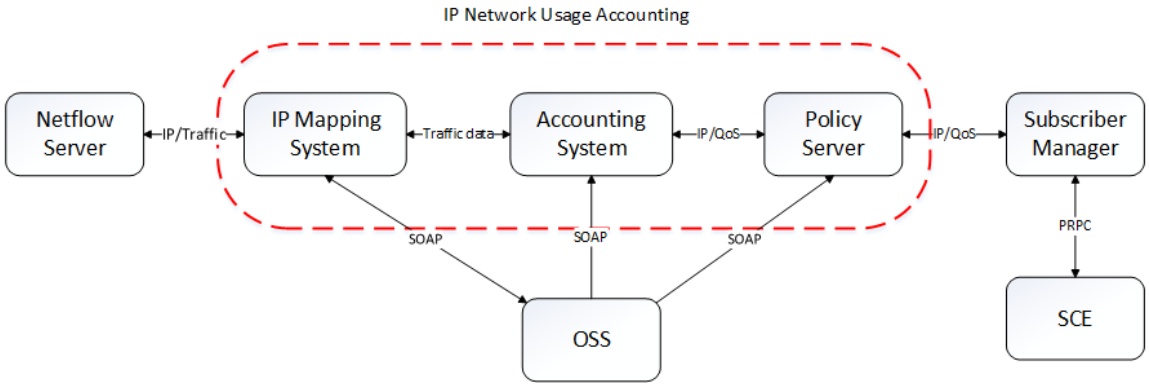

Figura 3 - Diagrama componente-conector do sistema

3.1.1 - *Accounting System*

O *Accounting System* é o subsistema central, responsável por manipular os registos recebidos do módulo *IP Mapping* e efetuar cálculos com o tráfego de cada cliente, de forma a chegar a um valor de QoS final que será depois enviado e atribuído pelo *Policy Server*. O sistema faz os cálculos com base nos seguintes parâmetros:

- Tráfego efetuado (indicado no registo que está a ser calculado no momento);
- Tráfego efetuado anteriormente (em todos os outros registos anteriores que já foram calculados);
- Período de tempo (data e hora da realização do tráfego);
- Produto do cliente (daqui em diante referido como *Access Type*), que indica quais as regras de negócio que serão usadas, ou por outras palavras, como serão feitos os cálculos e quais os valores QoS que é possível atribuir, segundo os resultados dos mesmos;

O *Accounting System* recebe dados do *IP Mapping* de 15 em 15 minutos, salvo erros do sistema. Esta periodicidade vem da própria recolha de dados pelo *Netflow*, que está limitado a recolhê-los de 15 em 15 minutos. Estes dados indicam, para um período de 15 minutos, qual a quantidade de tráfego em *bytes* que um determinado conjunto *MAC Address*/IP/Cliente efetuou, além de indicarem qual o produto, (*Access Type*) do cliente, necessário para que o *Accounting System* saiba que regras de negócio aplicar nos cálculos. É de notar que cada *MAC Address* ou CPE (*Customer Premises Equippment*) pode ter vários IP's, por exemplo, uma casa com um único CPE mas com vários computadores ligados à rede. Como tal, podem existir diversos registos com o mesmo *MAC Address* para um determinado período de tempo.

Para cada um desses dados, o *Accounting System* efetua diversos cálculos, aplicando as regras de negócio, para chegar a um valor final de tráfego ponderado e conforme esse valor e o produto do cliente, atribuir um QoS. Estas regras de negócio podem ser de três tipos:

- *Happy Hour* – Se o tráfego tiver sido feito numa determinada hora do dia, apenas conta uma certa percentagem (por exemplo, tráfego feito às 2h apenas conta 25%, logo de 1000 *bytes* de tráfego efetuado apenas contam 250 *bytes*);
- Desconto por *Access Type* – Semelhante à regra anterior, mas tem em conta o *Access Type* do cliente para aplicar o desconto (por exemplo, o tráfego feito às 2h apenas conta 25% se o *Access Type* for o 1, e o tráfego feito às 3h apenas conta 50% se o *Access Type* for o 5);
- Tráfego acumulado – Esta regra soma o tráfego feito entre dois períodos de tempo, aplica uma percentagem e devolve-o (por exemplo, o tráfego feito nas últimas 24h foi de 1000 *bytes* e o desconto é de 50%, logo a regra devolve 500 *bytes*);

Cada uma destas regras de negócio recebe um valor de tráfego, manipula-o conforme descrito acima, e devolve-o já com a percentagem aplicada. A primeira regra de negócio a ser aplicada recebe o valor de tráfego indicado no registo a ser processado, ou seja, a quantidade de tráfego feita em bruto naqueles 15 minutos. A segunda regra de negócio recebe o valor de tráfego já calculado pela primeira regra de negócio, e a terceira regra recebe o valor da segunda e por aí adiante.

As regras de negócio a aplicar são definidas pelo *Access Type.* Cada *Access Type* pode ter uma ou mais regras de negócio e cada regra de negócio pode ser aplicada em um ou mais *Access Types*. Embora possam existir várias instâncias da mesma regra de negócio, o que difere são os parâmetros das mesmas – período de tempo, percentagem, data em que a regra funciona, etc. A regra do tráfego acumulado, por exemplo, pode funcionar para o período das últimas 24h, da última semana, das últimas 3 semanas, ou até mesmo apenas das 24h anteriores até às 48h anteriores.

Cada instância de regra de negócio é identificada por um ID, e existe uma tabela que faz a associação *Access Type* – Regra de negócio e indica a ordem pela qual cada regra é aplicada.

Com o novo sistema *Accounting System* deverá ser possível criar novas regras de negócio, especificando os parâmetros, e associá-las a *Access Types*. No sistema anterior não havia esta possibilidade, sendo utilizadas sempre as mesmas regras, exceto a do desconto por *Access Type*.

O sistema também terá que permitir que certos IP's ou *MAC Addresses* sejam classificados como *overrides*. Isto significa que estes registos serão excluídos dos cálculos, atribuindo um QoS estático. Embora no sistema anterior já estivessem implementados os *overrides* por IP, pela primeira vez será possível fazê-lo também por *MAC Address*. Esta necessidade advém do facto de haver clientes que pagam valores mais elevados para terem sempre o mesmo QoS, independentemente do tráfego que efetuam.

3.2 - Definição de requisitos

Após várias reuniões com a NOS Madeira, foi possível chegar a um acordo em relação aos requisitos do sistema. Embora tenham havido várias alterações ao longo deste processo, o fio condutor do documento de requisitos manteve-se semelhante à primeira versão, sendo as reuniões seguintes necessárias apenas para acertar pequenos detalhes. Aqui é apresentada a versão final dos requisitos do sistema.

3.2.1 - Requisitos Funcionais

1) O sistema deve receber do *IP Mapping*, recorrendo ao protocolo HTTP, uma lista com os seguintes dados:
   a) IP;
   b) *MAC Address* do *modem*/CPE;
   c) Tráfego gerado;
   d) Período;
   e) Produto do cliente (*Access Type*);
   f) Número de cliente;
   g) Tipo de cliente;

2) Ao receber os dados, o sistema deve notificar o *IP Mapping* do sucesso ou da falha de receção, guardando os dados atomicamente ou não os guardando de todo, recorrendo a uma transação;
3) O sistema deve calcular sempre os dados mais antigos provenientes do *IP Mapping*;
4) O sistema deve enviar para o *Policy Server* um par IP/QoS por CPE;
5) O sistema deve suportar a operação de *pull* ao trocar dados com o módulo *Policy Server*;
6) O sistema deve suportar a operação de *push* ao trocar dados com o módulo *Policy Server*;
7) O sistema guardar o histórico de utilização de dados por *modem* (*MAC Address*) até 31 dias;
    a) Deve haver uma tabela com os dados de tráfego em bruto, inalterados, apenas para consulta;
    b) Deve haver uma tabela com os dados de tráfego já com as regras de negócio aplicadas;
        i) Deve haver a opção de eliminar completamente os dados desta tabela, através de *MAC Address*;
8) O sistema deve manter uma tabela de IP*/MAC Address*/QoS atribuídos correntemente;
9) O sistema deve possuir a capacidade de efetuar associação estática de QoS para determinados IPs de CPE, sobrepondo-se aos cálculos;
10) O sistema deve calcular as regras de negócio usando os seguintes parâmetros:
    a) Tráfego feito no registo a ser calculado;
    b) Histórico de tráfego do utilizador:
    c) Período de tempo;
    d) Produto do cliente;
11) O sistema deve disponibilizar como serviços todas as suas funcionalidades;
    a) Os serviços serão disponibilizados com recurso ao protocolo SOAP, com descrição em WSDL;
12) O sistema deve possuir uma interface gráfica para criar/eliminar regras de negócio;

3.2.2 - Requisitos Não Funcionais

1) O sistema deve ser implementado em PHP, com recurso à "*Zend Framework* 1.12";
2) O sistema deve implementar os serviços web com recurso ao protocolo SOAP, através do protocolo HTTP;
3) O sistema deve ser modular e ter acoplamento fraco;
4) O sistema deve efetuar um ciclo completo de processamento no tempo máximo de 15 minutos (considera-se um ciclo completo de processamento a partir do momento em que o sistema começa a processar uma lista do *IP Mapping* até ao momento em que envia a lista processada com os pares IP/QoS para o *Policy Server*);
5) O sistema deve distribuir o processamento por vários processadores;
6) O sistema deve ter um *uptime* de 99.999%;
7) O sistema deve ter um tempo de recuperação de 1 hora por cada 20 ciclos de processamento falhados;
8) O sistema deve ser desenvolvido com o suporte da metodologia de desenvolvimento "SCRUM";

3.3 - Arquitetura do sistema

Inicialmente, o sistema foi idealizado como um sistema dividido em vários módulos, com uma base de dados comum (Anexo I). Foi idealizado um módulo para cálculos, um para os *overrides*, um para as regras de negócio e um para estatística (mais tarde chamado de histórico). Durante o desenvolvimento, foi claro que os módulos pretendidos podiam ser todos resumidos num só, neste caso o das regras de negócio. Os restantes tornaram-se tabelas na base dados, sendo o comportamento dos mesmos apenas uma parte do código. Assim sendo:

- O módulo de *override* passou a ser implementado através de duas tabelas na base de dados, uma com a lista de IPs/sub-redes com *override*, e outra com os *MAC Address* com *override*. No código apenas se verifica se o tráfego que está sendo tratado está numa dessas duas tabelas, procedendo ao tratamento adequado do registo;
- O módulo de estatística passou a ser implementado através de apenas uma tabela na base de dados, responsável por guardar (até 30 dias) o resultado dos cálculos de tráfego efetuados para cada registo;
- O módulo dos cálculos passou a ser uma classe em PHP responsável por calcular todo o tráfego para os registos e guardar nas tabelas referidas nos pontos anteriores (além de outras tabelas não referidas aqui);
- O módulo de regras de negócio continuou a existir, mas com mais responsabilidades – todos os módulos idealizados nos pontos anteriores passaram a pertencer-lhe, além de uma mudança na filosofia: enquanto o idealizado era que este módulo tivesse informações sobre as regras de negócio, essas informações ficaram guardadas numa tabela da base de dados, sendo o módulo responsável por aplica-las corretamente;
- O módulo da interface gráfica continuou a existir da mesma forma, sendo responsável pela inserção e manutenção de regras de negócio e *overrides* na base de dados;
- Foram criados dois novos módulos – um para o *IP Mapping* e outro para o *Policy Server*, ambos responsáveis pelos serviços web entre o *Accounting System* e os dois sistemas referidos.

O diagrama final pode ser visto na Figura 4.

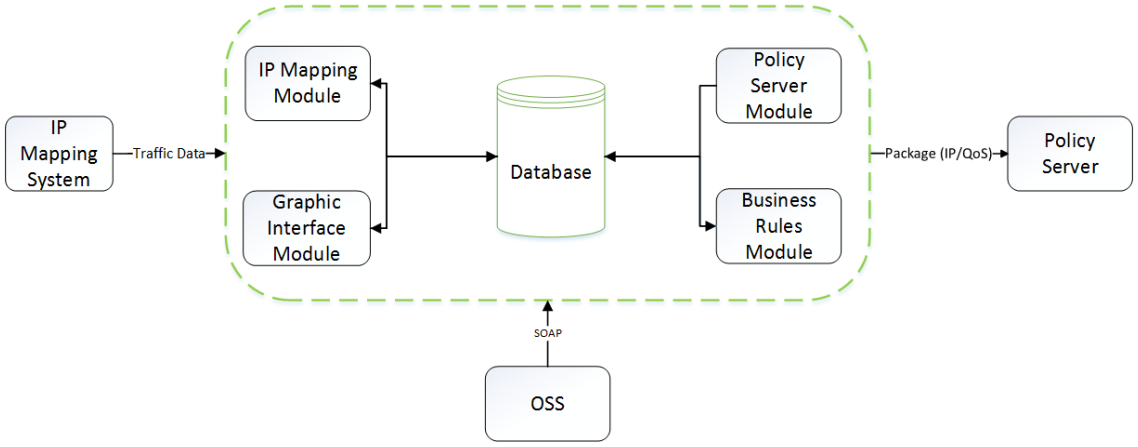

Figura 4 - Diagrama completo do Accounting System

3.4 - Diagrama de atividades

O diagrama de atividades foi crucial para entender o funcionamento do sistema, na fase inicial, e dividir a complexidade em várias operações que conduziriam ao resultado final. Logo de início foi possível observar que, tal como nos diagramas anteriores, a idealização era diferente da execução. Como se pode observar no Anexo II e comparando com a Figura 5, a noção de lista deixou de existir e foi substituída por "dados". Inicialmente, o sistema *Accounting* iria receber do *IP Mapping* uma lista de dados, processar a lista e notificar o *IP Mapping* que estava pronto para receber outra lista. Com o desenvolver do projeto, esta tarefa tornou-se completamente assíncrona, com o *Accounting* a receber dados em intervalos de tempo não especificados, e verificando periodicamente se tinha dados para processar.

Outra alteração prende-se com o módulo de estatística. Mesmo substituindo o módulo pela tabela na base de dados, a principal alteração é o facto de não haver verificação se é o primeiro cálculo de um determinado IP ou não. Simplesmente, existe uma regra de negócio que verifica qual o tráfego feito pelo IP num determinado período de tempo e devolve o valor acumulado.

Por último, em vez de apenas concatenar os *overrides* à tabela que é enviada ao *Policy Server*, o sistema também adiciona os ditos *overrides* às tabelas de histórico. Esta alteração vem da possibilidade de um IP ou *MAC Address* ser *override* numa determinada semana, deixar de o ser por uma semana, e posteriormente voltar a ser *override*. Da forma que o sistema estava planeado, durante o período em que o registo pertencia a um *override*, este não era registado na tabela de histórico. Embora existam mais *nuances* no sistema, este diagrama mostra de forma fácil de compreender o funcionamento geral do mesmo, sem entrar em detalhes desnecessários. É de notar que em cada ciclo de processamento de dados, as ações representadas neste diagrama correm paralelamente em várias instâncias, com cada processo a tratar de uma gama de dados distinta (por exemplo, o processo 1 trata dos primeiros 1000 dados, o processo 2 dos dados de 1001 a 2000, o processo 3 dos dados de 2001 a 3000, etc). O diagrama final pode ser visto na Figura 5.

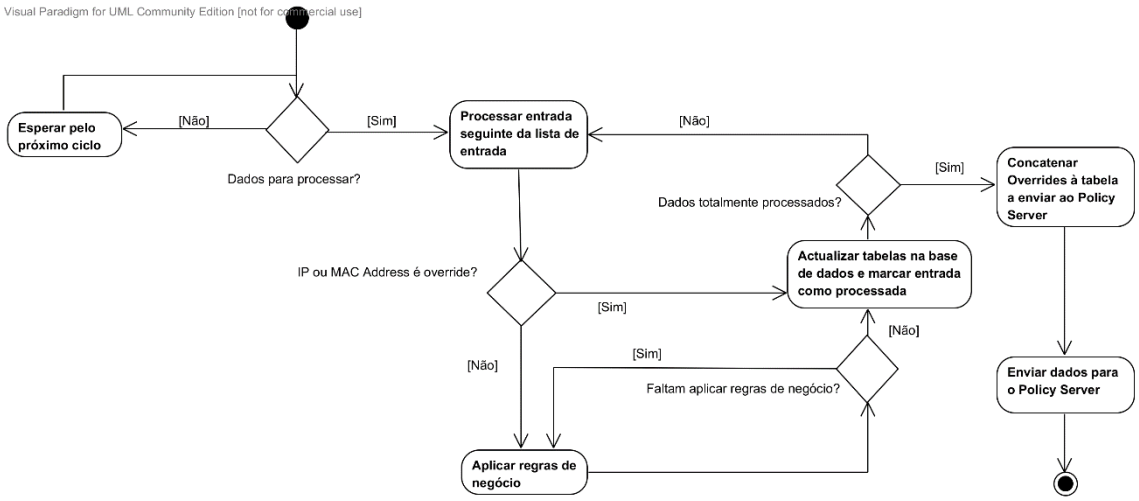

Figura 5 - Diagrama de atividades

3.5 - Metodologia de desenvolvimento – SCRUM

A metodologia de desenvolvimento usada no projeto foi o SCRUM. No SCRUM, o desenvolvimento de um produto ocorre em fases curtas, com o resultado de cada fase a incrementar o que foi feito na fase anterior. Um diagrama resumo pode ser visto na Figura 6.

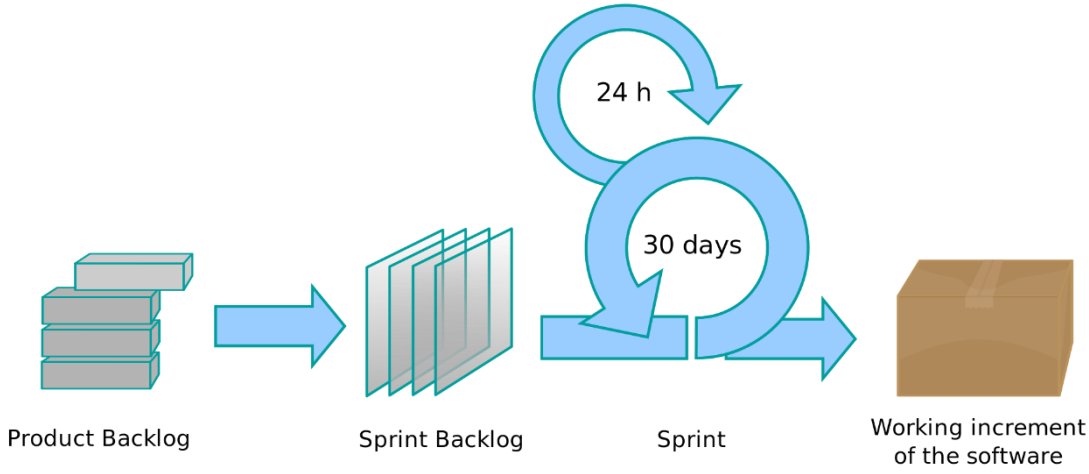

Figura 6 - Ciclo de desenvolvimento com SCRUM [23]

3.5.1 – Papéis no SCRUM

Nesta metodologia, existem três papéis principais [23]:

- *Product Owner* (cliente) – Decide o que tem que ser feito dentro dos próximos 30 dias;
- *Development Teams* (equipas de desenvolvimento) – Desenvolvem o que foi decidido pelo *product owner* em 30 dias ou menos, demonstrando no fim o que está feito. O *product owner* decide o que construir a seguir;
- *Scrum Master* (gestor do projeto) – Garante que o processo de desenvolvimento ocorre de forma eficaz, ajudando na melhoria do mesmo, da equipa e do produto.

3.5.2 – *Sprint*

O SCRUM é realizado através de *sprints*. Um *sprint* é um evento com a duração de um mês ou menos no qual é criado um incremento de um produto potencialmente final, pronto a vender. Um *sprint* começa imediatamente após o fim do *sprint* anterior. Podemos considerar cada *sprint* como um projeto com a duração de um mês.

Diariamente é feita uma reunião de 15 minutos, com o nome de *daily scrum*, em que os membros das equipas de desenvolvimento sincronizam as suas atividades e criam um plano para as 24h seguintes. Isto é feito analisando o trabalho que foi feito desde o último *daily scrum* e prevendo o trabalho que pode ser feito antes do próximo *daily scrum*. Nesta reunião,

que é feita todos os dias à mesma hora e no mesmo lugar para simplificar o processo, os membros de cada equipa de desenvolvimento explicam:

- O que fizeram no dia anterior que ajudou a equipa de desenvolvimento a aproximar-se do objetivo do *sprint*;
- O que vão fazer durante o dia para ajudar a equipa de desenvolvimento a aproximar-se do objetivo do *sprint*;
- Se há algum impedimento que pode impedir que o membro ou a equipa de desenvolvimento atinjam o objetivo do *sprint*.

O *scrum master* garante que a reunião ocorre, mas os membros da equipa de desenvolvimento são responsáveis por organizar-se durante a mesma [23].

3.5.3 – Artefactos do SCRUM

O SCRUM tem três artefactos principais para ajudar no desenvolvimento dos projetos, e garantir transparência de informação e compreensão dos mesmos.

O primeiro artefacto é o *product backlog*, que consiste numa lista ordenada de todas as necessidades do produto, sendo a única fonte de requisitos para quaisquer alterações que tenham que ser feitas ao mesmo. O *product owner* é responsável pelo *product backlog*, incluindo o seu conteúdo, disponibilidade e ordenação. Um *product backlog* nunca está completo: as primeiras versões apenas referem os requisitos conhecidos e mais bem compreendidos, evoluindo à medida que o projeto evolve para responder a novas necessidades.

O segundo artefacto é o *sprint backlog*. Este artefacto é o conjunto de itens do *product backlog* selecionados para o *sprint*, e representa que funcionalidades estarão presentes no próximo incremento do produto, além do trabalho necessário para colocar as ditas funcionalidades num produto potencialmente final.

O terceiro artefacto é o *increment* (incremento). Este artefacto é a soma de todos os itens do *product backlog* completos durante um *sprint* e o valor de todos os incrementos prévios. No fim de cada *sprint*, o novo incremento tem que estar no estado “feito”, o que significa que o produto tem que estar numa condição em que possa ser usado, independentemente do *product owner* querer ou não lançar o produto [23].

3.5.4 – SCRUM adaptado ao projeto

Neste projeto, o SCRUM não foi utilizado na íntegra, devido a algumas limitações incontornáveis:

- Havia uma única equipa de desenvolvimento, com apenas um membro;
- Tendo em conta o ponto anterior, não fazia sentido haver um *daily scrum* visto este ser realizado entre os diversos membros das equipas. Também era impossível ao

membro da equipa de desenvolvimento estar todos os dias presente na NOS Madeira para a reunião;
- O *scrum master* e o *product owner* eram representados pela mesma entidade, neste caso a NOS Madeira.

O *sprint backlog* não era um documento, mas sim um acordo sobre o que tinha que estar pronto em determinada data. A comunicação entre o desenvolvedor e o *product owner* era constante, embora não houvesse reuniões presenciais diárias, sendo substituídas por plataformas de mensagens instantâneas. Eram acertadas as funcionalidades que tinham que estar prontas, a data, e em cada um desses *sprints* o produto estava num estado potencialmente final, funcionando sem problemas, mas ainda sem todas as funcionalidades implementadas.

3.6 – Conclusão

Neste Capítulo foi abordada uma fase fulcral para o projeto: a modelação do sistema. Uma boa modelação num projeto de Engenharia de *Software* é o equivalente a uma boa planta de uma casa, feita pelo melhor arquiteto. Se a planta estiver errada, a casa será mal construída. Se a modelação estiver mal feita, o *software* terá diversos problemas. Esta analogia, embora simples em natureza, descreve perfeitamente a importância de planear cuidadosamente e com atenção ao detalhe um produto de *software*.

Na fase inicial, fez-se um resumo do objetivo do sistema como um todo, além dos objetivos individuais de cada módulo que o compõem. Imediatamente após, foi feito um levantamento do que já existe na infraestrutura da rede de suporte ao projeto, definindo como é que os componentes existentes comunicam com o sistema a ser desenvolvido. Isto originou o diagrama componente-conetor do sistema *IP Network Usage Accounting*.

De seguida, foi feita uma descrição textual detalhada do sistema a ser desenvolvido neste projeto, o subsistema *Accounting System*, que serve de referência sempre que for necessário resumir o seu funcionamento, sem entrar em detalhes técnicos.

Claro que para entender o funcionamento do sistema é antes necessário definir quais as suas funções. É aqui que se apresenta o documento de requisitos do sistema, que serviu de contrato entre o desenvolvedor do projeto e a NOS Madeira.

Com estes dados em mente, foi apresentada a arquitetura do subsistema *Accounting System*, seguido de um diagrama de atividades que exemplifica o funcionamento normal do sistema.

Finalmente, foi abordada a metodologia de desenvolvimento, neste caso o SCRUM. Embora não tenha sido cumprida na íntegra, a definição de uma metodologia é de extrema importância para ajudar a compreender o fluxo de trabalho e quais as expectativas do cliente e do desenvolvedor.

Terminada a modelação, segue-se o Capítulo 4, o mais extenso, trabalhoso e importante do projeto.

Capítulo 4 - Implementação

Neste capítulo serão descritos os módulos que compõem o sistema *Accounting*, com informações sobre o objetivo de cada um e explicação das decisões tomadas no código. Será dada a ênfase às dificuldades encontradas para dar uma noção de como o sistema evoluiu ao longo do projeto.

4.1 - Tecnologias utilizadas

Nesta secção serão descritas as tecnologias utilizadas para a realização do projeto.

4.1.1 - LAMP

O LAMP (*Linux, Apache, MySQL, PHP*) é um conjunto de *software* de código aberto bastante popular no desenvolvimento de aplicações web. Alguns dos componentes podem ser substituídos (mudando a designação do acrónimo). Por exemplo, Linux pode ser substituído por Windows e PHP pode ser substituído por Perl ou Python. A Google [24] e a Wikipedia [25] são duas empresas de grande nome que usam a *stack* LAMP nos seus websites.

4.1.2 - GIT

O GIT é um sistema distribuído de controlo de versões, ou seja, um sistema que guarda as alterações feitas a ficheiros para estes poderem ser acedidos nos seus estados anteriores no futuro [26]. A diferença entre GIT e outros sistemas do género, como por exemplo *Subversion* [27], é que enquanto estes sistemas guardam as alterações como diferenças ao longo do tempo entre os ficheiros (Figura 7), o GIT guarda o estado completo do sistema num género de "*snapshot*" (Figura 8). Em vez de voltar a guardar um ficheiro que não foi alterado, o GIT referencia a versão anterior do ficheiro [26].

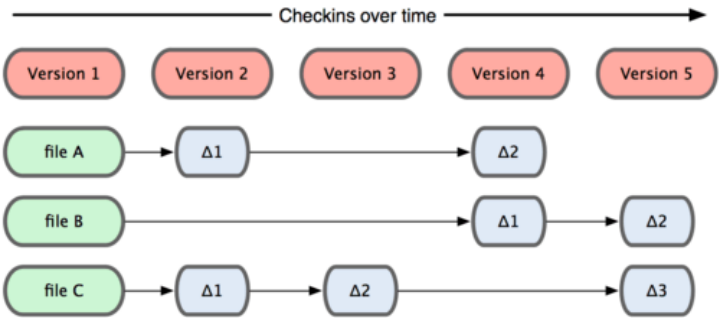

Figura 7 - Exemplo de checkins em Subversion [26]

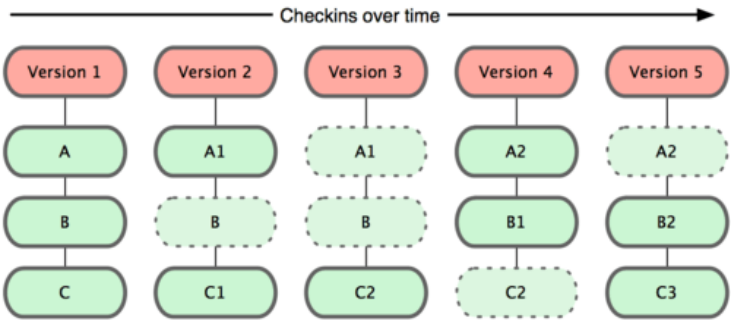

Figura 8 - Exemplo de checkins em GIT [26]

4.1.3 - *Zend Framework* 1.12

A *Zend Framework 1.12* foi escolhida para o projeto por ser a *framework* usada pela NOS Madeira. Segundo a documentação oficial, a *Zend Framework* é "uma *framework* em código aberto de aplicações web orientada por objetos para o PHP5. A *Zend Framework* é normalmente chamada de biblioteca de componentes, pois contém bastantes componentes com acoplamento fraco (*loose coupling*) que se podem usar mais ou menos independentemente" [28]. O acoplamento fraco implica que os componentes de um sistema têm pouco ou nenhum conhecimento das definições de outros componentes. No caso da *Zend Framework*, a vantagem desta característica reside no facto de ser possível aproveitar componentes da *framework* em aplicações web que não façam uso da mesma.

4.1.3.1 - *Model-View-Controller*

A *Zend Framework* faz uso do padrão de *software* MVC, ou *Model-View-Controller* (Modelo-Vista-Controlador). Este padrão separa uma aplicação em três partes interligadas (ver Figura 9), mas com responsabilidades diferentes [28]:

- ***Model*** – É a parte da aplicação que contém a funcionalidade básica;
- ***View*** – É a parte da aplicação que é apresentada ao utilizador.
- ***Controller*** – É a parte da aplicação que liga o *model* e a *view*. O *controller* manipula o modelo, e através dos pedidos do utilizador decide qual a *view* a apresentar.

Atualmente, a grande maioria das aplicações web faz uso do padrão MVC. A separação de responsabilidades leva a código mais legível e organizado, mais modularidade, maior facilidade de modificação e maior portabilidade.

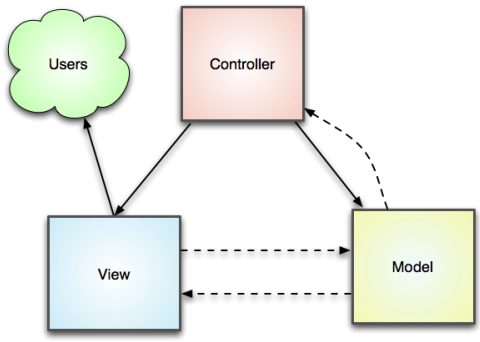

Figura 9 - Padrão MVC [28]

4.1.3.2 - *Auto-Discovery para WSDL da Zend Framework*

A *Zend Framework* facilita a criação dos ficheiros WSDL, permitindo ao programador preocupar-se apenas com a programação dos serviços disponíveis em código, sem ter que criar qualquer ficheiro XML. Isto é conseguido através de uma funcionalidade de *auto-discovery*. Para ativá-la, é necessário definir no controlador uma função "*wsdlAction*" tal como exemplificado no *Snippet* 5.

```
3  public function wsdlAction() {
4    $this->getHelper('ViewRenderer')->setNoRender(true);
5
6    $wsdl = new Zend_Soap_AutoDiscover();
7
8    $wsdl->setClass('Businessrules_Model_Services');
9
10   $wsdl->setUri('http://netaccounting.nosmadeira.pt/businessrules/index/soap');
11
12   $wsdl->setOperationBodyStyle(array('use' => 'literal'));
13
14   $wsdl->handle();
15 }
```

Snippet 5 - Definição do Auto-Discovery para WSDL

Com a função "*setClass*" indicamos qual o nome da classe em PHP que implementa os serviços disponibilizados. Nesta classe, cada função pública é um serviço, e para o WSDL ser gerado corretamente é necessário definir para cada uma um *docBlock* que indica quais os tipos de dados que a função recebe e retorna, conforme exemplificado no *Snippet* 6:

```
3  /**
4   * Gathers all processed traffic data from a given MAC Address
5   * @param string $macAddress
6   * @param string $startDate
7   * @param string $endDate
8   * @return object $result
9   */
10 public function trafficHistory($macAddress, $startDate, $endDate) {
11   //Código aqui
12 }
```

Snippet 6 - DocBlocks para gerar WSDL corretamente

Conforme o código do Snippet 6, o serviço *trafficHistory* recebe três parâmetros, todos eles *strings*, e retorna um objeto. O WSDL é automaticamente gerado conforme estes parâmetros. Se houver incompatibilidades entre o número de parâmetros que a função recebe/retorna e os *docBlocks*, o XML é mal formado e o WSDL fica inacessível.

4.2 - Base de dados

O sistema de bases de dados usado foi o MySQL. O motor usado foi o InnoDB sendo a escolha facilitada por um conjunto de fatores, já vistos na Secção 2.4.2:

- Suporta transações (ACID);
- Não bloqueia uma tabela onde esteja a ser feito um UPDATE ou DELETE;
- Recupera de falhas de forma bastante mais rápida e eficaz;
- Otimiza significativamente as *queries* que utilizam a chave primária da tabela;
- *Buffer Pool* do InnoDB acelera consideravelmente as *queries*;

Quanto à estrutura de tabelas da base de dados, esta foi adaptando-se às necessidades, à medida que a parte prática do projeto se materializou. Apesar do planeamento efetuado para evitar grandes alterações quando a base de dados começasse a armazenar dados, e embora houvesse uma ideia inicial de que dados seriam necessários guardar, e de quais as tabelas necessárias, houve diversas alterações que só foram implementadas em fases mais avançadas do projeto. A base de dados pode ser visualizada na Figura 10:

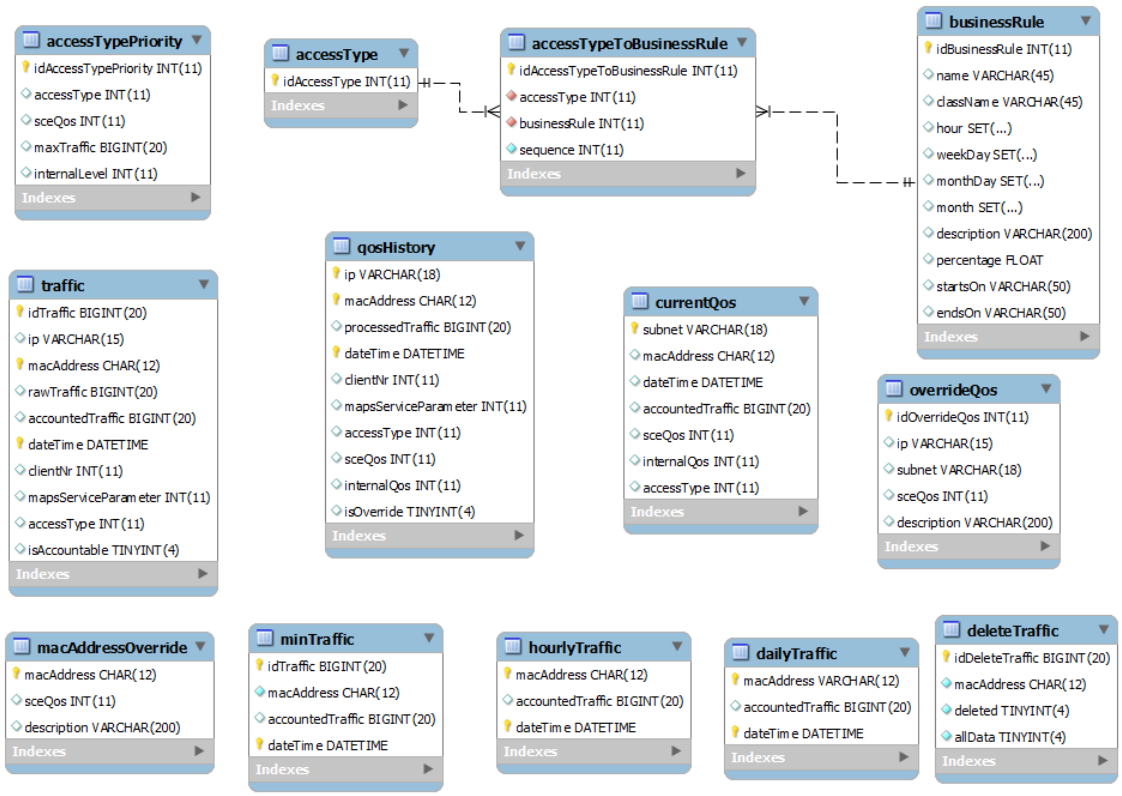

Figura 10 - Esquema da base de dados Accounting

4.2.1 – Tabelas

Nas seções seguintes serão descritas todas as tabelas que compõem a base de dados do subsistema *Accounting System*.

4.2.1.1 - Tabela Access Type

Esta tabela tem apenas um campo, *Id Access Type*. No momento em que este projeto foi feito, os *Access Types* na NOS Madeira eram identificados por um número, mas no futuro poderão ter um nome e/ou descrição. Como a tabela é de reduzidas dimensões, caso seja necessário fazer essa alteração não será necessário parar o sistema, pois a alteração será feita rapidamente. Para fácil identificação de cada produto, o *Id Access Type* corresponde sempre ao número usado pela NOS Madeira para identificar os *Access Types*.

4.2.1.2 - Tabela Business Rule

Esta tabela contém, além do ID e do nome, todos os outros parâmetros necessários para definir as regras de negócio e torna-las extensíveis e modulares:

- ***Class Name*** – Define o nome da classe em PHP que contém o código necessário para executar a regra de negócio. Esta classe faz uso dos restantes parâmetros da tabela para funcionar corretamente;
- ***Hour*, *Week Day*, *Month Day*, *Month*** – Respetivamente, hora, dia da semana (de segunda a sexta), dia do mês e mês. As regras aplicam-se nas horas e dias marcados;
- ***Percentage*** – Percentagem aplicada ao tráfego calculado com a regra;
- ***Starts On* e *Ends On*** – Servem para as regras que vão buscar o tráfego entre dois períodos de tempo indicando, em minutos, qual o período inicial e o período final (por exemplo, 1440 minutos correspondem a um dia e 2880 minutos a dois dias. Uma regra com estes parâmetros ia buscar o tráfego feito 24h antes até 48h antes e aplicar a percentagem);

4.2.1.3 - Tabela Access Type to Business Rule

Esta tabela faz a associação entre os *Access Types* e as regras de negócio. Cada *Access Type* pode ter uma ou mais regras de negócio, e uma regra de negócio pode ser aplicada em um ou mais *Access Types*. O campo *sequence* indica a ordem pela qual as ordens são aplicadas em cada *Access Type*.

4.2.1.4 - Tabela Access Type Priority

Esta tabela define quais os níveis de prioridade para cada *Access Type*:

- ***Sce QoS*** – Indica o pacote de QoS que é aplicado no SCE;
- ***Internal Level*** – Indica o nível interno de cada *Access Type*;
- ***Max Traffic*** – Indica o valor máximo de tráfego, em *bytes*, que um IP pode fazer para estar dentro de um determinado nível;

4.2.1.5 - Tabela MAC Address Override

Esta tabela armazena os *MAC Address* que são *override*, indicando qual o seu pacote no SCE e uma descrição de a quem pertence o *MAC Address*.

4.2.1.6 - Tabela Override QoS

Esta tabela armazena os *overrides* por IP, indicando não apenas o IP mas também qual a sub-rede a que cada um pertence, e qual o seu pacote no SCE.

4.2.1.7 - Tabela Traffic

É nesta tabela que os dados provenientes do *IP Mapping* são guardados. Guarda os dados de tráfego até 30 dias. Além dos campos "auto explicáveis", como o *Id Traffic*, IP, *MAC Address*, *Date Time* e C*lient Nr* (número de cliente) existem quatro outros que merecem especial atenção:

- ***Raw Traffic*** - É o tráfego feito em bruto pelo IP, representado em *bytes*, no período de tempo indicado no campo *Date Time*;
- ***Accounted Traffic*** – É o tráfego feito já com o desconto da regra da *Happy Hour*;
- ***Maps Service Parameter*** – Campo não utilizado até à data de conclusão do projeto, mas criado a pedido da NOS Madeira caso seja necessário no futuro, evitando que seja necessário pausar o processamento do sistema quando hipoteticamente se adicionasse o campo à tabela com 119 milhões de registos. Representa o produto do cliente para fins de faturação, ao contrário do *Access Type* que é o produto usado para definir os cálculos;
- ***Is Accountable*** – É um registo do tipo *bit* (1 ou 0) que indica se o registo já foi processado ou não. É a partir deste campo que o módulo das regras de negócio verifica se há dados para processar ou não.

  Esta tabela implementa o requisito funcional 7.a.

4.2.1.8 - Tabelas Min Traffic, Hourly Traffic e Daily Traffic

Inicialmente, estas tabelas não existiam. Sempre que era necessário ir buscar dados de tráfego, a tabela *Traffic* era usada. No entanto, à medida que a tabela foi crescendo, as regras de negócio que a utilizavam começaram a afetar significativamente o desempenho do sistema. A título de exemplo, a regra *Time Weight*, que calcula a soma do tráfego feito entre dois

períodos de tempo, tinha que por vezes percorrer a tabela toda para somar registos. A tabela *traffic* tem, em média, 115 200 000 registos a cada momento. Isto porque existem, em média, 40 000 clientes ativos e, para cada IP, existem 4 registos por hora, vezes 24 horas, vezes 30 dias, vezes 40 000 clientes. Mesmo com índices bem delineados e após a implementação de partições, o desempenho continuava fraco.

Foi, então, proposta a ideia de criar três tabelas de resumo (ver secção 5.1.2). A tabela *Min Traffic* guarda apenas os dados das últimas 24 horas. A tabela *Hourly Traffic* guarda os dados desde a hora anterior até uma semana antes, mas com uma *nuance*: em vez de 4 registos por hora, tem apenas um registo por hora para cada *MAC Address*. Por último, a tabela *Daily Traffic* guarda os dados de tráfego desde as 00h do dia corrente até 30 dias antes, sendo que cada *MAC Address* tem um único registo para cada dia, com o tráfego somado.

A tabela *Min Traffic* guarda os seus dados ao mesmo tempo que estes são inseridos na tabela *Traffic*. A tabela *Hourly Traffic* gera os dados através de um *cron* que corre de hora a hora e agrupa todo o tráfego feito por todos os *MAC Addresses* até 24h antes, indo buscar os dados à tabela *Min Traffic*. A tabela *Daily Traffic* gera os dados através de um *cron* que corre 5 minutos depois da meia-noite, agrupando o tráfego feito nos últimos dias por cada *MAC Address,* num período de 24h. Para isso, busca os dados à tabela *Hourly Traffic*. É muito mais rápido ir buscar os dados às tabelas resumo que à tabela *Traffic* no momento da geração de dados, pois a quantidade de registos a agrupar é menor.

Embora se perca a granularidade de ter os dados de 15 em 15 minutos, a diferença nos cálculos é insignificante quando comparada com os ganhos no desempenho. Esta perda de granularidade acontece porque por exemplo para cada *MAC Address*, se o sistema quiser encontrar qual o tráfego que foi feito há exatamente 14 dias, às 12h30m, não encontra essa informação, encontrando no seu lugar um único registo para o tráfego completo feito pelo *MAC Address* nesse dia. Estas tabelas implementam o requisito funcional 7.b.

4.2.1.9 - Tabela QoS History

Esta tabela indica o valor final do tráfego calculado, qual o pacote no SCE e nível interno de cada registo e se este era *override* ou não no momento dos cálculos.

4.2.1.10 - Tabela Current QoS

Esta é a tabela que contém os valores atuais de tráfego processado, pacote no SCE e nível interno para cada *MAC Address/Subnet.* Esta tabela contém, em média, 40 000 registos, e é da mesma que partem os dados para o *Policy Server*.

4.2.1.11 - Tabela Delete Traffic

Esta tabela guarda os dados sobre quais os *MAC Addresses* que deverão ver o seu tráfego ser apagado, com um campo para indicar se a operação já foi ou não efetuada. O

campo *All Data* indica se é para apagar todos os dados relativos ao *MAC Address* ou apenas os dados que são usados nas regras de negócio futuras.

4.2.2 - Partições

Uma das soluções implementadas para melhorar o desempenho do sistema foi a criação de partições. Estas foram feitas por dia, para cada uma das tabelas de tráfego: *Min Traffic, Hourly Traffic, Daily Traffic, Traffic* e *QoS History*. As *queries* são mais eficientes quando contêm um WHERE com data, pois o MySQL procura automaticamente numa determinada partição e ignora todas as outras. A sintaxe base para criar as partições pode ser vista no *Snippet* 7:

```
3  alter table accounting.traffic
4  partition by range (TO_DAYS(dateTime)) (
5      PARTITION p20140822 VALUES LESS THAN (TO_DAYS('2014-08-23')),
6      PARTITION p20140823 VALUES LESS THAN (TO_DAYS('2014-08-24')),
7      PARTITION p20140824 VALUES LESS THAN (TO_DAYS('2014-08-25')),
8      (...),
9      PARTITION p20140924 VALUES LESS THAN (TO_DAYS('2014-09-25')),
10     PARTITION p20140925 VALUES LESS THAN (TO_DAYS('2014-09-26'))
11 )
```

Snippet 7 - Criação de partições

4.2.3 - Replicação

Segundo a documentação oficial do MySQL, a replicação é um processo que permite que os dados de um servidor de base de dados MySQL, o *master* (mestre) sejam copiados para um ou mais servidores de base de dados MySQL, os *slaves* (escravos) [29]. Esta cópia é assíncrona, permitindo que possam existir pausas ou falhas de comunicação entre os *slaves* e o *master*, sem que se percam dados. As vantagens da replicação e, consequentemente, as razões que levaram a que a mesma fosse implementada no projeto são as seguintes:

- Embora a escrita de dados tenha que ser feita sempre no *master*, a leitura pode ser distribuída por um ou mais *slaves*, balanceando a carga para aumentar o desempenho;
- Existe uma maior segurança de dados visto ser possível, por exemplo, pausar a replicação num *slave* e correr um serviço de *backup* no mesmo, enquanto o *master* continua a receber dados;
- No caso concreto do projeto, um dos objetivos foi haver a possibilidade de parar o servidor *master* completamente, enquanto um dos *slaves* pode assumir o seu papel (ou seja, assumir o papel de *master* e também a responsabilidade pelo recebimento de dados do *IP Mapping*, dos cálculos, do envio de dados para o *Policy Server* e dos serviços web disponibilizados ao OSS) até o *master* voltar a funcionar. A este processo dá-se o nome de replicação passiva. Desta forma é possível fazer manutenção nas máquinas sem ser necessário pausar o funcionamento do subsistema.

A Figura 11 mostra um exemplo de um processo de replicação. Todos os comandos que o *master* faz/recebe são escritos no *Binary Log*. Por sua vez, o *slave* ao conectar-se ao *master*, recebe o *Binary Log*. O *slave* lê o ficheiro e aplica nas suas bases de dados as mesmas alterações que foram feitas no *master*.

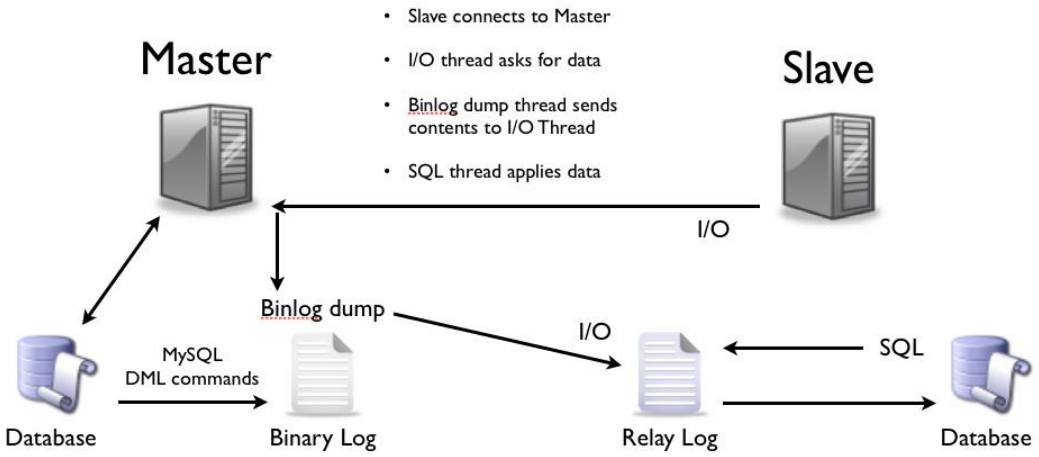

Figura 11 - Exemplo de um processo de replicação

4.3 – Módulo *IP Mapping*

Este módulo é responsável por toda a comunicação com o sistema *IP Mapping* (tendo o mesmo nome que o dito subsistema). A sua única responsabilidade é fornecer um serviço web que recebe os dados do *IP Mapping*, guarda-os na base de dados e retorna uma notificação de sucesso ou falha. Durante quase todo o processo de desenvolvimento, este módulo apenas disponibilizou o serviço *Push Bulk*. O diagrama de classes do módulo pode ser visto na Figura 12.

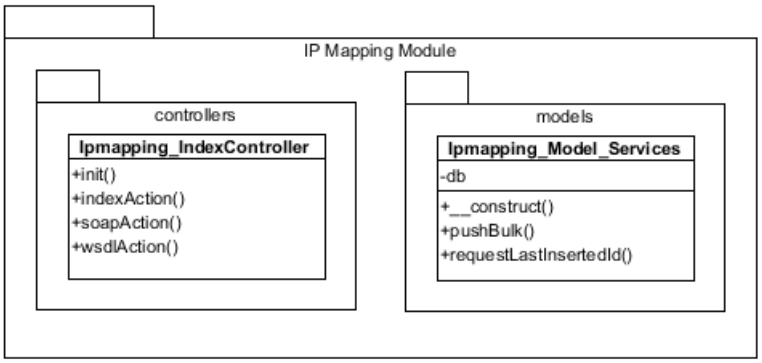

Figura 12 - Diagrama de classes do módulo IP Mapping

4.3.1 - Função *Push Bulk*

Este é o serviço que recebe os dados do *IP Mapping* e guarda-os na base de dados. Recebe um *array* de dados e retorna um booleano – TRUE para sucesso, FALSE para erro.

A solução implementada foi construir um único INSERT com todos os registos, conforme exemplificado no *Snippet 8*.

```
3  $sql = "INSERT IGNORE INTO traffic (idTraffic, ip, macAddress, rawTraffic,
4          accountedTraffic, dateTime, clientNr, accessType) VALUES ";
5
6  $sqlMinute = "INSERT IGNORE INTO minTraffic (idTraffic, macAddress,
7                accountedTraffic, dateTime) VALUES ";
8
9  foreach($rawData as $row => $value) {
10     $sql .= "('$value[0]', '$value[1]', '$value[2]', '$value[3]',
11             '$value[3]', '$value[4]', '$value[5]', '$value[6]'),";
12
13     $sqlMinute .= "('$value[0]', '$value[2]',
14                   '$value[3]', '$value[4]'),";
15 }
16
17 //Remove the last comma
18 $sql = substr($sql, 0, -1);
19 $sqlMinute = substr($sqlMinute, 0, -1);
```

Snippet 8 - Um único INSERT com todos os dados

É de notar que são construídas duas *queries*, uma para a tabela *Traffic* e outra para a tabela *Min Traffic*. Após vários testes de desempenho, a diferença média de tempo entre inserir apenas numa tabela ou inserir nas duas rondou os 5 segundos. Devido a problemas com a geração de tráfego para a tabela *Min Traffic* na solução prévia, que recorria a um script externo executado após cada inserção na tabela *Traffic*, foi decidido deixar a geração dos dados nesta função. Os dados são inseridos nas duas tabelas recorrendo a transações, o que impede que existam dados numa tabela que não estejam presentes na outra. Esta função implementa os requisitos funcionais 1 e 2.

4.3.2 - Função *Request Last Inserted Id*

Como os dados que o *IP Mapping* envia para o *Accounting* têm como chave primária um ID e, devido a problemas de reset ao *auto-increment* da coluna, foi implementada a função *Request Last Inserted Id* do lado do *Accounting.* Esta função verifica qual ID máximo inserido na tabela *Traffic* e retorna-o, para que os dois sistemas estejam sempre sincronizados (ver *Snippet* 9).

```php
/**
 * RequestLastInsertedId method
 * Returns the last inserted id from the traffic table
 * @return int $result
 */
public function requestLastInsertedId() {
    $sql = "SELECT MAX(idTraffic) FROM traffic";
    $result = $this->db->fetchOne($sql);
    return $result;
}
```

Snippet 9 - Função/Serviço Request Last Inserted Id

4.4 – Módulo *Policy Server*

O módulo *Policy Server* (tem o mesmo nome que o subsistema com o qual comunica), à semelhança do *IP Mapping*, é um módulo de reduzidas dimensões. Apenas fornece serviços web ao sistema *Policy Server* para verificar o QoS atual por *MAC Address* e por IP. O diagrama de classes do módulo pode ser visto na Figura 13.

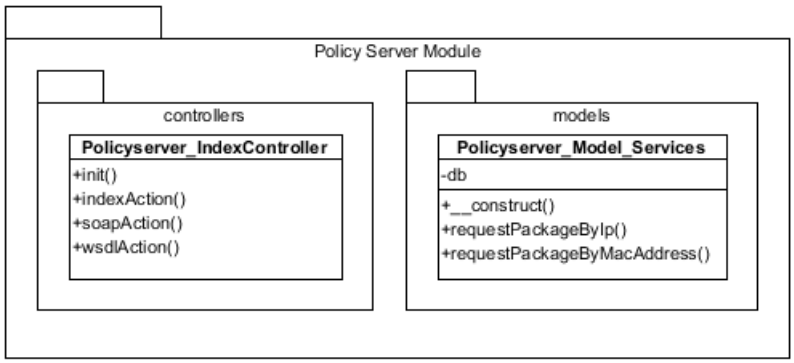

Figura 13 - Diagrama de classes do módulo Policy Server

4.4.1 - Função *Request Package By IP*

Esta função verifica na tabela de QoS atual a existência do IP, concatenado com "/32", e o seu QoS. Esta particularidade advém do facto dos dados na tabela serem todos endereços IP de sub-redes e não simplesmente IPs. Uma sub-rede /32 é, na realidade, um IP ou *host* único, daí concatenarmos o valor /32 na *query*. Caso não sejam encontrados registos, é feita uma *query* à tabela de *overrides* para verificar a existência de QoS estático para o registo (ver *Snippet* 10). Esta função implementa o requisito funcional 5.

```php
/**
* Verifies the priority of a given IP
* @param string $ip
* @return object $pair
*/
public function requestPackageByIp($ip) {
    $sql = "SELECT subnet as ip, sceQos as package FROM currentQos WHERE subnet LIKE '$ip/32'";
    $pair = $this->db->fetchRow($sql, null, Zend_DB::FETCH_OBJ);
    if(!$pair) {
        $sql = "SELECT subnet as ip, sceQos as package FROM overrideQos WHERE ip LIKE '$ip'";
        $pair = $this->db->fetchRow($sql, null, Zend_DB::FETCH_OBJ);
    }
    return $pair;
}
```

Snippet 10 - Função/Serviço Request Package By Ip

4.4.2 - Função *Request Package By MAC Address*

Esta função é semelhante à anterior, mas faz a verificação de QoS atual através de *MAC Address* em vez de IP (ver *Snippet* 11). Esta função implementa o requisito funcional 5.

```php
/**
* Verifies the priority of a given MacAddress
* @param string $macAddress
* @return int $priority
*/
public function requestPackageByMacAddress($macAddress) {
    $sql = "SELECT sceQos AS package FROM currentQos WHERE macAddress = $macAddress";
    $priority = $this->db->fetchOne($sql);
    return $priority;
}
```

Snippet 11 - Função/Serviço Request Package By MAC Address

4.5 - Módulo *Business Rules*

Este é o módulo principal do sistema. É o módulo que implementa as regras de negócio, que efetua os cálculos de tráfego e que fornece os serviços web ao Departamento de Sistemas de Informação da empresa, respondendo às necessidades dos operadores e dos clientes quando é necessário consultar dados. Ao contrário dos dois módulos anteriores, que apenas continham um ficheiro/classe relevantes, este módulo contém 6 classes e a complexidade das mesmas é significativamente superior à das classes dos outros módulos. O diagrama de classes do módulo pode ser visto na Figura 14.

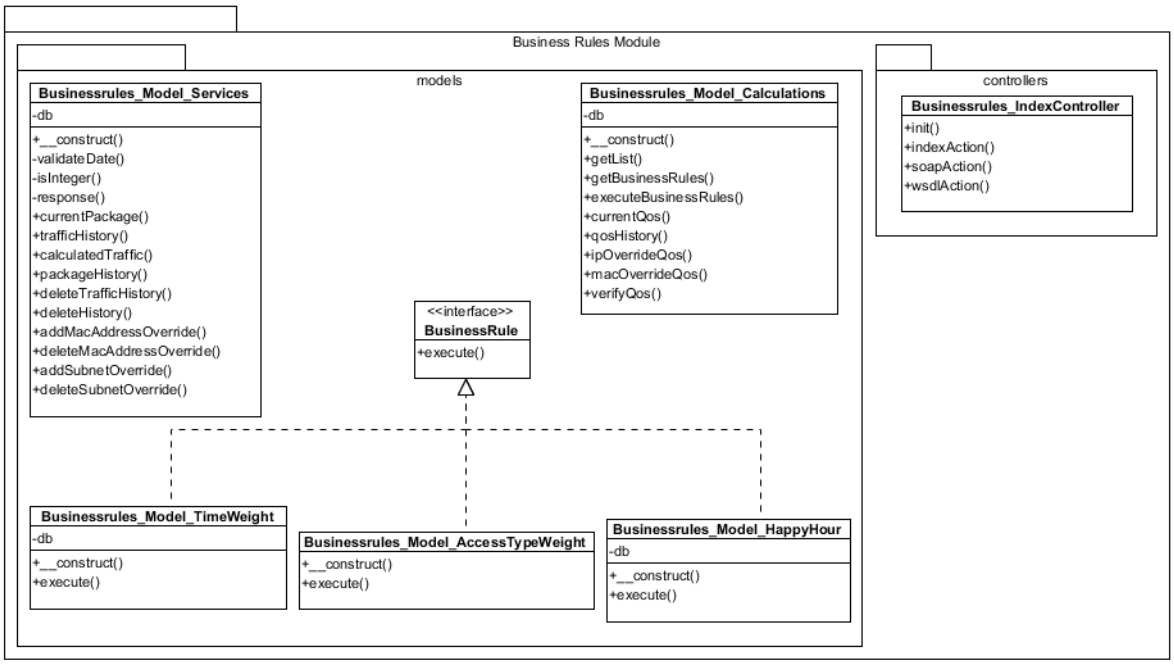

Figura 14 - Diagrama de classes do módulo Business Rules

4.5.1 - Classe *Calculations*

Esta é a classe principal do módulo. Percorre os registos um a um, aplicando as regras de negócio coincidentes com os seus *Access Types* e guardando os resultados na base de dados.

4.5.1.1 - Função Get List

Este é o ponto de partida para os cálculos. Esta função seleciona todos os registos da tabela *Traffic* que ainda não foram calculados, num intervalo definido pelos dois parâmetros que recebe: "$lowerLimit" e "$upperLimit". O primeiro parâmetro indica qual o primeiro registo usado, e o segundo indica quantos registos serão usados. Os dois parâmetros são inseridos na *query* na cláusula LIMIT. Deste modo, esta função pode ser chamada várias vezes de forma paralela, com parâmetros diferentes, para calcular diferentes conjuntos de registos. Como a função é chamada várias vezes ao mesmo tempo, e para evitar que uma dessas instâncias escolha os mesmos dados que outra instância é feito um *sleep* de 8 segundos a seguir à escolha dos mesmos (*Snippet* 12). Esta função implementa o requisito funcional 3.

```
3   public function getList($lowerLimit, $upperLimit) {
4       $sql = "SELECT * FROM traffic
5               WHERE isAccountable IS NULL
6               LIMIT $lowerLimit , $upperLimit";
7           $list = $this->db->fetchAll($sql);
8           sleep(8);
9           $this->getBusinessRules($list);
10  }
```

Snippet 12 – Função Get List

4.5.1.2 - Função Get Business Rules

Esta função é responsável por controlar o fluxo de processamento de cada registo, verificando se o IP ou *MAC Address* que está a verificar no momento é *override* e chamar as funções necessárias para o funcionamento do projeto. Recebe como parâmetro a lista de registos proveniente da função *Get List*.

A primeira operação da função é colocar numa *string* todos os IPs que são *overrides* e noutra *string* todos os *MAC Addresses* que são *overrides*. Estes registos ficam separados por uma barra "|". No *Snippet* 13 podemos ver como é feita esta operação para os IPs (os *MAC Addresses* são feitos de forma bastante semelhante):

```
3   $sqlOverrides = "SELECT ip FROM overrideQos";
4   $overrideList = $this->db->fetchAll($sqlOverrides);
5   $overrides = "_|";
6   foreach($overrideList as $row => $value) {
7       $overrides .= $value['ip'] . "|";
8   }
```

Snippet 13 - Overrides por IP numa única string, separados por "|"

Inicialmente, estas *queries* não existiam e cabia à função *Get List* separar os IPs e *MAC Addresses* que eram *override* com uma cláusula NOT IN, ou seja, na *query* "**SELECT * FROM traffic WHERE isAccountable IS NULL LIMIT $lowerLimit, $upperLimit**" acrescentávamos um "**AND ip NOT IN (SELECT ip FROM overrideQos)**" e um "**AND macAddress NOT IN (SELECT macAddress FROM macAddressOverride)**". No entanto, desta forma não era possível fazer qualquer operação com os registos que eram *overrides*, o que depois se revelou ser errado. Por outro lado, o desempenho sofria imenso com as listas a demorar muito para serem geradas, sendo bastante mais rápido colocar numa única *string* os *overrides* e depois decidir o que fazer.

De seguida, a função coloca em dois *arrays* todos os registos das tabelas *Access Type To Business Rule* e *Access Type Priority*, conforme exemplificado no *Snippet* 14:

```
3   $sqlBusinessRules = "SELECT * FROM accessTypeToBusinessRule ORDER BY SEQUENCE";
4   $businessRules = $this->db->fetchAll($sqlBusinessRules);
5
6   $sqlPriority = "SELECT * FROM accessTypePriority ORDER BY maxTraffic";
7   $priority = $this->db->fetchAll($sqlPriority);
```

Snippet 14 - Dados das regras de negócio num só array

O passo seguinte da função *Get Business Rules* é iterar sobre a lista de dados que tem para processar, verificando logo de início se o registo corrente é um *override* ou não. O IP e *MAC Address* do registo são comparados com as *strings* já construídas dos *overrides*, usando a função *strpos* do PHP. Se a função devolver FALSE, então, o registo é *override* (*Snippet* 15):

```
3  foreach ($list as $row => $traffic) {
4      $boolOverride = strpos($overrides, "|" . $traffic['ip'] . "|");
5      $boolMacOverride = strpos($macOverrides, "|" . $traffic['macAddress'] . "|");
```

Snippet 15 - Verificação de overrides

Cada registo é tratado de forma atómica, recorrendo a transações. Mesmo que o processamento dê erro num determinado registo, o sistema continua a funcionar, passando ao registo seguinte. Após a verificação dos *overrides*, são efetuadas várias chamadas às outras funções da classe necessárias para continuar os cálculos (*Snippet* 16). É de notar que, se o registo a ser tratado no momento for *override*, o processo é semelhante mas não são aplicadas as regras de negócio, ou seja, as funções E*xecute Business Rules* e *Verify QoS* não são usadas. No fim do processamento, é chamada a função *IP Override QoS.*

```
3  $this->db->beginTransaction();
4  if(!$boolOverride && !$boolMacOverride) {
5      try {
6          $accountedTraffic = $this->executeBusinessRules($businessRules, $traffic);
7          $qos = $this->verifyQos($traffic, $accountedTraffic, $priority);
8          $this->currentQos($traffic, $accountedTraffic, $qos);
9          $this->qosHistory($traffic, $accountedTraffic, $qos);
10         $this->db->update('traffic',
11                 array('isAccountable' => 1),
12                 'idTraffic = ' . "$traffic[idTraffic]");
13         $this->db->commit();
```

Snippet 16 - Chamadas às funções para calcular e guardar dados na BD

4.5.1.3 - Função Execute Business Rules

Esta é a função que verifica quais as regras de negócio que o *Access Type* do registo a ser processado no momento implementa. O tráfego inicial é o tráfego que o registo indica ter feito, sendo atribuído à variável A*ccounted Traffic*. De seguida, o sistema percorre os registos no *array* definido na função anterior, com os dados da tabela A*ccess Type To Business Rule.* Como esses registos estão ordenados pelo campo *sequence*, basta verificar se o *Access Type* do *array* equivale ao A*ccess Type* do registo que está a ser processado. Se sim, basta ir buscar as regras de negócio à base de dados e através do campo *Class Name* invocar a classe que contém a implementação da regra de negócio. Qualquer uma destas classes devolverá o valor do tráfego calculado até ao momento, sendo que a variável *Accounted Traffic* está sempre a ser atualizada com cada chamada às regras de negócio. A função pode ser vista no *Snippet* 17. Esta função implementa o requisito funcional 7.a.i.

```php
public function executeBusinessRules($businessRules, $traffic) {
    $accountedTraffic = $traffic['rawTraffic'];
    foreach ($businessRules as $row => $businessRule) {
        if($businessRule['accessType'] == $traffic['accessType']) {
            $sql = "SELECT * FROM businessRule "
                 . "WHERE idBusinessRule = $businessRule[businessRule]";
            $result = $this->db->fetchRow($sql);
            $class = new $result['className'];
            $accountedTraffic = $class->execute($traffic, $result, $accountedTraffic);
        }
    }
    return $accountedTraffic;
}
```

Snippet 17 - Função Execute Business Rules

4.5.1.4 - Função Verify QoS

Esta função é fulcral no sistema, pois é aqui que se verifica qual o QoS que será atribuído a cada registo, através do valor final de tráfego calculado. Primeiro, são definidas duas variáveis para o QoS – uma para o QoS que vai para o SCE, e outra para o QoS interno da regra de negócio. Estas duas variáveis são inicializadas a 1. De seguida, o sistema percorre o *array* que contém os dados da tabela A*ccess Type Priority*. Como indicado na secção da base de dados, esta tabela indica para cada *Access Type* qual o valor máximo de tráfego que um cliente pode fazer para obter um determinado QoS. Ao percorrer o *array,* é feita uma verificação para saber se o *Access Type* do *array* equivale ao A*ccess Type* do registo que está a ser processado. Como os registos estão ordenados pelo valor do tráfego máximo possível para cada nível, o primeiro valor encontrado é o que contém o melhor QoS (menos tráfego feito). Imediatamente, atribui-se às variáveis de QoS os valores indicados e depois é feita uma verificação: se o valor de tráfego máximo indicado no registo do *array* for superior ao valor de tráfego efetuado, então o QoS atribuído está correto e é feito um *break* para interromper a execução do *foreach*. Caso contrário, o sistema verifica o registo seguinte do *array* até encontrar o correto. Se nunca encontrar, o valor de QoS é 1 por defeito tanto para o SCE como a nível interno. O código da função está no *Snippet* 18:

```php
public function verifyQos($traffic, $accountedTraffic, $priority) {
    $qos = 1;
    $internalQos = 1;
    foreach ($priority as $row => $value) {
        if($value['accessType'] == $traffic['accessType']) {
            $qos = $value['sceQos'];
            $internalQos = $value['internalLevel'];
            if ($value['maxTraffic'] >= $accountedTraffic) {
                break;
            }
        }
    }
    $qosArray = array('qos' => $qos,
                      'internalQos' => $internalQos);
    return $qosArray;
}
```

Snippet 18 - Função VerifyQoS

4.5.1.5 - Função Current QoS

Esta função guarda na tabela *Current QoS* o valor atual de QoS para cada IP/*MAC Address*/cliente. Pode haver mais que um *MAC Address* na tabela, mas os IPs são únicos. Se o registo tiver uma chave duplicada (IP), é feito um "ON DUPLICATE KEY UPDATE" que permite atualizar os valores do QoS, a hora, o tráfego contabilizado e até mesmo os *Access Types* e *MAC Address*, pois estes também podem mudar (o *Access Type* muda se o cliente mudar de produto e o *MAC Address* que pertence ao CPE pode ir para outro cliente e obter outro IP). O código da função está no *Snippet* 19. Esta função implementa o requisito funcional 8.

```
3  public function currentQos($traffic, $accountedTraffic, $qos) {
4      $currentQos = "INSERT INTO currentQos (subnet, macAddress,
5                     sceQos, dateTime, accountedTraffic, internalQos, accessType) "
6                   . "VALUES ('$traffic[ip]', '$traffic[macAddress]', '$qos[qos]',
7                      NOW(), $accountedTraffic, '$qos[internalQos]', '$traffic[accessType]') "
8                   . "ON DUPLICATE KEY UPDATE "
9                   . "sceQos = '$qos[qos]', "
10                  . "internalQos = '$qos[internalQos]', "
11                  . "accessType = '$traffic[accessType]', "
12                  . "macAddress = '$traffic[macAddress]',"
13                  . "dateTime = NOW(), "
14                  . "accountedTraffic = $accountedTraffic";
15
16     $this->db->getConnection()->exec($currentQos);
17 }
```

Snippet 19 - Função Current QoS

4.5.1.6 - Função QoS History

Esta é uma das funções mais simples de toda a classe: apenas guarda na tabela do histórico os dados do tráfego processado (*Snippet* 20).

```
3  public function qosHistory($traffic, $accountedTraffic, $qos) {
4      $qosHistory = array(
5          'processedTraffic' => $accountedTraffic,
6          'ip' => $traffic['ip'],
7          'macAddress' => $traffic['macAddress'],
8          'dateTime' => $traffic['dateTime'],
9          'sceQos' => $qos['qos'],
10         'mapsServiceParameter' => $traffic['mapsServiceParameter'],
11         'accessType' => $traffic['accessType'],
12         'clientNr' => $traffic['clientNr'],
13         'internalQos' => $qos['internalQos']
14     );
15
16     $this->db->insert('qosHistory', $qosHistory);
17 }
```

Snippet 20 - Função QoS History

4.5.1.7 - Função IP Override QoS

Esta função é chamada após todos os registos estarem calculados para concatenar as sub-redes que são *override* à tabela de QoS atual (*Snippet* 21). Esta função implementa o requisito funcional 9.

```
3   public function ipOverrideQos() {
4       $sql = "INSERT INTO currentQos (subnet, sceQos, dateTime) "
5            . "SELECT DISTINCT(subnet), sceQos, NOW() "
6            . "FROM overrideQos "
7            . "ON DUPLICATE KEY UPDATE "
8            . "dateTime = NOW()";
9       $this->db->getConnection()->exec($sql);
10  }
```

Snippet 21 - Função IP Override QoS

4.5.1.8 - Função MAC Override QoS

Esta função é chamada sempre que um registo é identificado como sendo *override* por *MAC Address*. Como a tabela *MAC Address Override* não tem IPs, ao contrário da *Override QoS*, é necessário tratar do registo imediatamente (*Snippet* 22). Esta função implementa o requisito funcional 9.

```
3   public function macOverrideQos($traffic) {
4       $traffic['ip'] .= "/32";
5
6       $sqlQos = "SELECT sceQos FROM macAddressOverride "
7               . "WHERE macAddress = '$traffic[macAddress]'";
8       $qos = $this->db->fetchOne($sqlQos);
9
10      $currentQos = "INSERT INTO currentQos (subnet, macAddress, sceQos, dateTime) "
11                  . "VALUES ('$traffic[ip]', '$traffic[macAddress]', $qos, NOW()) "
12                  . "ON DUPLICATE KEY UPDATE "
13                  . "sceQos = $qos, "
14                  . "macAddress = '$traffic[macAddress]',"
15                  . "dateTime = NOW()";
16      $this->db->getConnection()->exec($currentQos);
17  }
```

Snippet 22 - Função MAC Override QoS

4.5.2 - Interface *Business Rule*

Foi criada uma interface para as regras de negócio. Cada classe que a implemente terá uma função *execute* que recebe como parâmetros os dados de tráfego, os dados da regra de negócio respetiva e o valor de tráfego contabilizado. A interface pode ser vista no *Snippet* 23. As classes que estendem esta interface implementam o requisito funcional 10.

```php
interface BusinessRule {

    public function execute($traffic, $businessRule, $accountedTraffic);

}
```

Snippet 23 - Interface das regras de negócio

4.5.3 - Classe Businessrules Model Happy Hour

Esta classe implementa a regra de negócio da *Happy Hour*, que aplica uma percentagem ao tráfego feito se este tiver sido realizado numa determinada hora e data, indicada nos campos *Hour, Week Day, Month Day* e *Month*. É de notar que, mesmo que não esteja definido um dia da semana, se o tráfego for feito num valor que está indicado no campo dos dias do mês, então, o desconto também é aplicado, e vice-versa (isto supondo que a hora e mês também batem certo). Esta classe também atualiza o campo A*ccounted Traffic* nas tabelas *Traffic* e *Min Traffic*, para estarem sempre atualizados para regras de negócio futuras. A implementação da classe está demonstrada no *Snippet* 24.

```php
public function execute($traffic, $businessRule, $accountedTraffic) {
    $dateTime = new DateTime($traffic['dateTime']);
    $hour = explode(',', $businessRule['hour']);
    $weekDay = explode(',', $businessRule['weekDay']);
    $monthDay = explode(',', $businessRule['monthDay']);
    $month = explode(',', $businessRule['month']);
    if (in_array($dateTime->format('n'), $month)
        && (in_array($dateTime->format('j'), $monthDay)
        || in_array($dateTime->format('N'), $weekDay))
        && in_array($dateTime->format('G'), $hour)) {

        $accountedTraffic *= $businessRule['percentage'];
        $this->db->update('traffic',
            array('accountedTraffic' => $accountedTraffic), 'idTraffic = ' . "$traffic[idTraffic]");
        $this->db->update('minTraffic',
            array('accountedTraffic' => $accountedTraffic), 'idTraffic = ' . "$traffic[idTraffic]");
    }
    return $accountedTraffic;
}
```

Snippet 24 - Implementação da regra de negócio Happy Hour

4.5.4 - Classe Businessrules Model Access Type Weight

Tal como a regra *Happy Hour*, esta regra de negócio aplica um desconto se o tráfego for feito num determinado período de tempo. A diferença é que não atualiza os campos *Accounted Traffic* e a percentagem aplicada está relacionada com o *Access Type*. A implementação está no *Snippet* 25.

```php
public function execute($traffic, $businessRule, $accountedTraffic) {
    $dateTime = new DateTime($traffic['dateTime']);
    $hour = explode(',', $businessRule['hour']);
    $weekDay = explode(',', $businessRule['weekDay']);
    $monthDay = explode(',', $businessRule['monthDay']);
    $month = explode(',', $businessRule['month']);
    if (in_array($dateTime->format('n'), $month)
        && (in_array($dateTime->format('j'), $monthDay)
        || in_array($dateTime->format('N'), $weekDay))
        && in_array($dateTime->format('G'), $hour)) {

        $accountedTraffic *= $businessRule['percentage'];
    }
    return $accountedTraffic;
}
```

Snippet 25 - Implementação da regra de negócio Access Type Weight

4.5.5 - Classe Businessrules Model Time Weight

Esta classe implementa a regra de negócio mais complexa. O objetivo é agarrar no tráfego feito num período de tempo relativo à data do registo, por exemplo, as 24h anteriores ou de 24h antes até uma semana antes, somar e aplicar uma percentagem. Esta foi a regra que levou à criação das 3 tabelas de tráfego, pois usando apenas a tabela *Traffic* o processamento tornou-se incrivelmente lento. As tabelas podem ter registos em comum (por exemplo, a tabela *Min Traffic* tem registos das últimas 24h, e a tabela *Hourly Traffic* tem das últimas 23h, entre outras). Claro que, ao calcular o tráfego, não se pode usar tráfego repetido, por isso a solução é para cada tabela apenas aceder a registos que já tenham passado uma determinada data (*Hourly Traffic* – 1440 minutos ou 24h e *Daily Traffic* – 10080 minutos ou 7 dias). Com esta verificação, a *query* faz uma consulta às 3 tabelas para verificar se existem dados para o período de tempo especificado na definição da regra de negócio. O tráfego é somado e, no fim, aplicada a percentagem. A implementação da regra de negócio pode ser vista no *Snippet* 26.

```
3  public function execute($traffic, $businessRule, $accountedTraffic) {
4
5      $sql = "SELECT SUM(accountedTraffic), macAddress FROM (
6              SELECT macAddress, SUM(accountedTraffic) AS accountedTraffic
7              FROM minTraffic
8              WHERE macAddress = '$traffic[macAddress]'
9              AND dateTime BETWEEN DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[endsOn])
10             AND DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[startsOn])
11             AND dateTime >= DATE_SUB('$traffic[dateTime]', INTERVAL 1440 MINUTE)
12             GROUP BY macAddress
13             UNION
14             SELECT macAddress, SUM(accountedTraffic) AS accountedTraffic
15             FROM hourlyTraffic
16             WHERE macAddress = '$traffic[macAddress]'
17             AND dateTime BETWEEN DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[endsOn])
18             AND DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[startsOn])
19             AND dateTime < DATE_SUB('$traffic[dateTime]', INTERVAL 1440 MINUTE)
20             GROUP BY macAddress
21             UNION
22             SELECT macAddress, SUM(accountedTraffic) AS accountedTraffic
23             FROM dailyTraffic
24             WHERE macAddress = '$traffic[macAddress]'
25             AND dateTime BETWEEN DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[endsOn])
26             AND DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[startsOn])
27             AND dateTime < DATE_SUB('$traffic[dateTime]', INTERVAL 10080 MINUTE)
28             GROUP BY macAddress ) x GROUP BY macAddress";
29     $result = $this->db->fetchOne($sql);
30     if($result > 0) {
31         $accountedTraffic += $result * $businessRule['percentage'];
32     }
33     return $accountedTraffic;
34 }
```

Snippet 26 - Implementação da regra de negócio Time Weight

4.5.6 - Classe Businessrules Model Services

A última classe do módulo das regras de negócio contém todos os serviços web disponibilizados aos Sistemas de Informação. Nesta secção nem sempre será apresentado o código completo de cada função, sendo que nas funções maiores optou-se por explicar apenas alguns detalhes mais relevantes.

4.5.6.1 - Função privada Response

Para estar conforme os padrões dos serviços web dos Sistemas de Informação, cada serviço tem que devolver um objeto com dois atributos: um que indique o código de erro (0 se não houver erro) e qual o texto do erro (ou outra mensagem a indicar que a operação foi bem sucedida). Estes dois atributos estão presentes nas respostas de todos os serviços, independentemente de outros atributos que a resposta envie. A função *response* serve para gerar um objeto conforme especificado. A implementação está no *Snippet* 27.

```php
/**
 * Standard function for the structure used in all SOAP replies
 * @param int $errorId
 * @param string $errorText
 * @return object $object
 */
private function response($errorId, $errorText) {
    $object = new stdClass();
    $object->errorId = $errorId;
    $object->errorText = $errorText;
    return $object;
}
```

Snippet 27 - Função response para resposta padrão

4.5.6.2 - Função privada Is Integer

Devido às necessidades de alguns serviços de validar certos valores como sendo inteiros, foi criada uma função que verifica se o parâmetro recebido é inteiro ou não, retornando TRUE ou FALSE conforme o resultado. A implementação está no *Snippet* 28.

```php
/**
 * Function that verifies if a given value is a positive integer
 * @param int $value
 * @return boolean
 */
private function isInteger($value) {
    if(!is_numeric($value) || $value < 0 || $value != round($value))
        return false;
    else
        return true;
}
```

Snippet 28 - Função Is Integer

4.5.6.3 - Função privada Validate Date

A função *Validate Date* recebe como parâmetro uma *string* que usa para criar um objeto *Date Time*. Se conseguir criar o objeto e o formato da *string* for igual a "Ano-Mês-Dia Hora:Minuto:Segundo" (em PHP: "Y-m-d H:i:s") a data é válida. Esta função é chamada quando algum serviço usa uma ou mais datas como parâmetros. A implementação está no *Snippet* 29.

```php
/**
 * Function that verifies if a date is in the correct format
 * @param date $date
 * @param string $format
 * @return boolean
 */
private function validateDate($date, $format = 'Y-m-d H:i:s')
{
    $d = DateTime::createFromFormat($format, $date);
    return $d && $d->format($format) == $date;
}
```

Snippet 29 - Função Validate Date

4.5.6.4 - Função Current Package

Esta função devolve o QoS (*package*) atual de um determinado *MAC Address*. A implementação está no *Snippet* 30.

```php
/**
 * Returns the current package of a given MAC Address
 * @param string $macAddress
 * @return object $result
 */
public function currentPackage($macAddress) {
    if(strlen($macAddress)!= 12) {
        return $this->response(1, 'Invalid MACAddress length');
    }

    $sql = "SELECT sceQos, internalQos, accountedTraffic FROM currentQos WHERE macAddress = '$macAddress'";
    $currentPackage = $this->db->fetchOne($sql);
    if(!$currentPackage) {
        return $this->response(2, 'MACAddress not found');
    }

    $result = $this->response(0, 'OK');
    $result->currentPackage = $currentPackage;
    return $result;
}
```

Snippet 30 - Função Current Package

4.5.6.5 - Função Calculated Traffic

Esta função retorna, para um determinado *MAC Address*, o tráfego que foi feito nas últimas 24h, na última semana e nas últimas 3 semanas, sem haver tráfego repetido em qualquer um dos três períodos de tempo. A *query* em SQL está no *Snippet* 31.

```
3  $sql = "SELECT (SELECT SUM(accountedTraffic) FROM minTraffic
4          WHERE macAddress = '$macAddress' AND dateTime BETWEEN
5          DATE_SUB(NOW(), INTERVAL 1440 MINUTE) AND DATE_SUB(NOW(), INTERVAL 0 MINUTE)
6          GROUP BY macAddress) AS lastDay,
7          (SELECT SUM(accountedTraffic) FROM hourlyTraffic
8          WHERE macAddress = '$macAddress' AND dateTime BETWEEN
9          DATE_SUB(NOW(), INTERVAL 10080 MINUTE) AND DATE_SUB(NOW(), INTERVAL 1440 MINUTE)
10         GROUP BY macAddress) AS lastWeek,
11         (SELECT SUM(accountedTraffic) FROM dailyTraffic
12         WHERE macAddress = '$macAddress' AND dateTime BETWEEN
13         DATE_SUB(NOW(), INTERVAL 30240 MINUTE) AND DATE_SUB(NOW(), INTERVAL 10080 MINUTE)
14         GROUP BY macAddress) AS lastThreeWeeks";
```

Snippet 31 - Função Calculated Traffic

4.5.6.6 - Função Package History

A função *Package History* retorna os valores de QoS (*packages*) de um determinado *MAC Address* entre dois períodos de tempo (*query* no *Snippet* 32. Se não houver dados correspondentes na tabela do histórico, é feita uma nova *query* (*Snippet* 33) para verificar qual foi o valor de QoS atribuído mais recentemente antes do período inicial especificado. Se ainda assim não forem encontrados dados, a função retorna "*No data found*".

```
3  $sql = "SELECT sceQos, internalQos, dateTime, isOverride "
4         . "FROM qosHistory WHERE macAddress = '$macAddress' "
5         . "AND dateTime BETWEEN '$startDate' AND '$endDate'";
6
7  $packageHistory = $this->db->fetchAll($sql);
```

Snippet 32 - Query para verificar o QoS atribuído num período de tempo

```
3  if(empty($packageHistory)) {
4      $sql = "SELECT sceQos, internalQos, dateTime, isOverride "
5             . "FROM qosHistory WHERE macAddress = '$macAddress' "
6             . "AND dateTime = (SELECT MAX(dateTime) "
7             . "FROM qosHistory WHERE macAddress = '$macAddress' "
8             . "AND dateTime <= '$startDate')";
9      $packageHistory = $this->db->fetchAll($sql);
10
11     if(empty($packageHistory))
12         return $this->response(3, "No data found");
13 }
```

Snippet 33 - Query caso não sejam encontrados dados no período especificado

4.5.6.7 - Função Traffic History

Esta função retorna o tráfego em bruto efetuado por um *MAC Address* entre dois períodos de tempo. Após validar as datas, o sistema coloca na variável $*startDate* o campo dos segundos a 00 e o campo dos minutos a 00, 15, 30 ou 45 (*Snippet* 34). Por norma, estes campos já vêm corretos, mas se, por exemplo, uma das datas indicar "2014-09-01 14:37:12", na realidade o registo que procuramos na base de dados é "2014-09-01 14:35:00". Esta transformação é necessária devido a um dos passos seguintes da função.

```php
$dateTime = new DateTime($startDate);
$hour = $dateTime->format('H');
$minute = $dateTime->format('i');

if ($minute > 0 && $minute <= 15 ) {
    $dateTime->setTime($hour, 15, 00);
}
else if ($minute > 15 && $minute <= 30 ) {
    $dateTime->setTime($hour, 30, 00);
}
else if ($minute > 30 && $minute <= 45 ) {
    $dateTime->setTime($hour, 45, 00);
}
else {
    $dateTime->setTime($hour, 00, 00);
}
$startDate = $dateTime->format('Y-m-d H:i:s');
```

Snippet 34 - Transformação da data inicial na função Traffic History

De seguida, é feita a consulta à base de dados para verificar se existem registos entre os períodos indicados. A *query* está no *Snippet* 35.

```php
$sql = "SELECT SUM(accountedTraffic) AS accountedTraffic, dateTime "
    . "FROM traffic WHERE macAddress = '$macAddress' "
    . "AND dateTime BETWEEN '$startDate' AND '$endDate' GROUP BY dateTime";
```

Snippet 35 - SQL para buscar dados entre duas datas

Com o *array* de dados já em memória, o sistema percorre-o e compara o valor da data do registo com uma variável que contém o valor da data inicial fornecida, e que a cada iteração é aumentado em 15 minutos. Se os registos forem iguais, significa que existe tráfego naquela data/hora, e o registo é incluído no *array* final. Se não houver registo, é criado um com a data que está atribuída na variável que comparamos, e com o tráfego a 0. Este registo é inserido no *array* final. Este ciclo continua enquanto a variável com a data for maior que a data de fim especificada nos parâmetros.

Inicialmente, se um *MAC Address* só tivesse feito tráfego às 15:00 e depois às 15:45, por exemplo, não havia qualquer registo a indicar 0 tráfego feito às 15:30. Devido às necessidades dos Sistemas de Informação, tornou-se necessário fazer esta verificação. Esta é também a razão principal pela qual é necessário tratar a data de início logo no início da função, pois só assim conseguimos adicionar 15 minutos à variável que faz a comparação no *while*, de forma a ter os registos corretos. O código para esta secção está no *Snippet* 36.

```php
$finalArray = array();
$count=0;
while($startDate <= $endDate) {
    $dateString = $startDate->format('Y-m-d H:i:s');

    if(@$trafficHistory[$count]['dateTime'] == $dateString) {
        array_push($finalArray, $trafficHistory[$count]);
        $count++;
    }
    else {
        $dateInsert = $startDate->format('Y-m-d H:i:s');
        array_push($finalArray, array('accountedTraffic' => '0', 'dateTime' => "$dateInsert"));
    }
    $startDate->add(new DateInterval('PT15M'));
}
```

Snippet 36 - Ciclo para verificar se existem registos a cada 15 minutos

4.5.6.8 - Função MAC Address Override

Esta função adiciona um *override* por *MAC Address* à base de dados (*Snippet* 37).

```php
$this->db->insert('macAddressOverride', array(
                    'macAddress' => $macAddress,
                    'sceQos' => $qos,
                    'description' => $description));

return $this->response(0, 'Record inserted');
```

Snippet 37 - SQL para inserir override por MAC Address na base de dados

4.5.6.9 - Função Add Subnet Override

Esta função adiciona uma sub-rede à lista de *overrides*, além dos IPs individuais que fazem parte da sub-rede. Após usar a função do PHP *ip2long* para verificar que o IP é válido, verificamos se a máscara de sub-rede se encontra entre 1 e 32 e é um valor inteiro (*Snippet* 38).

```php
if(!ip2long($subnet)) {
    return $this->response('2', 'Invalid IP/subnet');
}
if(!$this->isInteger($mask) || $mask > 32 || $mask < 1) {
    return $this->response('3', 'Invalid subnet mask');
}
```

Snippet 38 - Validação da sub-rede e da máscara

De seguida, é criado um *array* onde são colocados todos os IPs que pertencem à sub-rede (*Snippet* 39). Por último, inserimos os registos na base de dados.

```
$addresses = array();
if (($min = ip2long($subnet)) !== false) {
    $max = ($min | (1<<(32-$mask))-1);
    for ($i = $min; $i <= $max; $i++)
      $addresses[] = long2ip($i);
}
```

Snippet 39 - Geração de IPs da sub-rede

4.5.6.10 - Funções Delete MAC Address Override e Delete Subnet Override

Estas duas funções fazem exatamente aquilo que o nome indica: apagar os *overrides* por *MAC Address* e por sub-rede.

4.5.6.11 - Função Create Business Rule

Esta função cria novas instâncias de regras de negócio de acordo com os parâmetros especificados, após validação dos mesmos.

4.5.6.12 - Função Delete Traffic History

Esta função recebe como parâmetro um *MAC Address* cujos registos na base de dados devem ser apagados apenas nas tabelas que armazenam os dados usados nos cálculos (*Min Traffic, Hourly Traffic* e *Daily Traffic*). Primeiro, a função verifica se já existe algum pedido para apagar dados para aquele *MAC Address* em fila de espera, representada pela tabela *Delete Traffic* e, se existir, retorna mensagem a indicar que a eliminação de dados já está agendada (*Snippet* 40):

```
$sql = "SELECT idDeleteTraffic FROM deleteTraffic
        WHERE macAddress = '$macAddress'
        AND deleted = '0'";
$isDeleted = $this->db->fetchOne($sql);

if($isDeleted > 0) {
    $result = $this->response(0, "MAC Address '$macAddress' is already scheduled for deletion");
    $result->idDelete = $isDeleted;
    return $result;
}
```

Snippet 40 - Verificação de existência de pedido para apagar tráfego

Caso a *query* anterior não devolva qualquer registo, a função guarda na tabela *Delete Traffic* um registo com um ID, o *MAC Address*, o campo *Deleted* a 0 (indicando que ainda não foi apagado) e o campo *All Data* a 0 (indicando que é para apagar apenas nas três tabelas referidas acima). A função/serviço retorna, então, o ID que ficou armazenado na tabela (*Snippet* 41):

```
3  $data = array('macAddress' => "$macAddress", 'allData' => '0');
4  $this->db->insert('deleteTraffic', $data);
5  $id = $this->db->lastInsertId();
6
7  $result = $this->response(0, "MAC Address '$macAddress' traffic data scheduled for deletion");
8  $result->idDelete = $id;
9  return $result;
```

Snippet 41 - Agendando a eliminação dos dados na base de dados

4.5.6.13 - Função Delete History

Esta função/serviço é bastante semelhante à função *Delete Traffic History*, sendo a única diferença que aqui o objetivo não é apenas eliminar o tráfego usado para os cálculos, mas também todos os outros registos associados ao *MAC Address* fornecido. A *query* inicial para verificar se já há agendamento de eliminação de dados tem uma pequena diferença, verificando qual o tipo de eliminação, com o campo *All Data*. (*Snippet* 42):

```
3  $sql = "SELECT idDeleteTraffic, allData FROM deleteTraffic
4          WHERE macAddress = '$macAddress'
5          AND deleted = '0'";
6  $isDeleted = $this->db->fetchRow($sql);
```

Snippet 42 - Verificação de existência de dados a eliminar para o MAC Address

Logo de seguida e ao contrário da função/serviço *Delete Traffic History*, são feitas duas verificações. A primeira verifica se existe algum registo pendente para eliminar todos os dados para o *MAC Address* fornecido e, se sim, retorna o ID da operação. A segunda verificação verifica se existe algum registo pendente para eliminar os dados de tráfego do *MAC Address* fornecido onde o campo *All Data* esteja a 0. Se sim, atualiza o campo para 1 e retorna o ID da operação. Caso contrário, apenas insere o novo registo na base de dados. O código está no *Snippet* 43:

```
3  if($isDeleted['idDeleteTraffic'] > 0 AND $isDeleted['allData'] == 1) {
4      $result = $this->response(0, "MAC Address '$macAddress' already scheduled for deletion.");
5      $result->idDelete = $isDeleted['idDeleteTraffic'];
6      return $result;
7  }
8
9  if($isDeleted['idDeleteTraffic'] > 0 AND $isDeleted['allData'] == 0) {
10     $this->db->update('deleteTraffic',
11                       array('allData' => '1'),
12                       'idDeleteTraffic = ' . $isDeleted['idDeleteTraffic']);
13     $result = $this->response(0, "MAC Address '$macAddress' updated to delete all data.");
14     $result->idDelete = $isDeleted['idDeleteTraffic'];
15     return $result;
16 }
```

Snippet 43 - Verificações na função Delete History

4.5.7 - Scripts

Para minimizar a carga no servidor *Apache*, melhorar o desempenho e permitir que código pudesse ser executado periodicamente, via *crons* do Unix, foram criados diversos ficheiros de *scripts*, que correm o seu PHP fora do *Apache*. Todos estes *scripts* têm em comum a inicialização da aplicação *Zend Framework* para poder usar os recursos da mesma, como o *bootstrap* e a conexão à base de dados. Esta última é útil para podermos usar comandos da base de dados da *framework.* O código descrito está no *Snippet* 44.

```
3  define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/../../application'));
4
5  define('APPLICATION_ENV',
6      (getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') : 'development'));
7
8  require_once "Zend/Application.php";
9
10 $application = new Zend_Application(
11     APPLICATION_ENV,
12     APPLICATION_PATH . '/configs/application.ini'
13 );
14
15 $application->bootstrap();
16
17 $db = Zend_Db::factory('Pdo_Mysql', array(
18     'username' => 'root',
19     'password' => '123456',
20     'dbname'   => 'accounting'
21 ));
22
23 date_default_timezone_set('Europe/Lisbon');
```

Snippet 44 - Código comum a todos os scripts

4.5.7.1 - Script Main

Este script é executado de 4 em 4 minutos pelo *cron* da máquina e é o que dá inicio ao ciclo de processamento de dados. A sua primeira ação é tentar adquirir um recurso (um ficheiro de texto) em modo exclusivo através da função *fopen* do PHP. Se conseguir, pode continuar o processamento. Caso contrário, significa que outro processo já está na posse do recurso e o *script* para imediatamente. Apesar do ficheiro estar na mesma pasta que os *scripts*, é necessário especificar o caminho completo do mesmo pois, caso fosse usado o caminho relativo, ao correr o *script* pelo *cron* a função não ia conseguir encontrar o ficheiro corretamente (*Snippet* 45).

```
3  $lock = fopen('/var/vhost/netaccounting/application/scripts/lockMain.txt', 'w');
4  if(!flock($lock, LOCK_EX | LOCK_NB)) {
5      exit();
6  }
```

Snippet 45 - Adquirindo um recurso para poder iniciar o processamento

De seguida, o *script Main* chama o *script Calculations* várias vezes, mas com parâmetros diferentes, conforme se pode ver no *Snippet* 46.

```
3  $cmd  = "php " . APPLICATION_PATH . "/scripts/Calculations.php 0 2499  & ";
4  $cmd .= "php " . APPLICATION_PATH . "/scripts/Calculations.php 2499 2499 & ";
5  $cmd .= "php " . APPLICATION_PATH . "/scripts/Calculations.php 4998 2499 & ";
6  //...
7  //...
8  $cmd .= "php " . APPLICATION_PATH . "/scripts/Calculations.php 34986 2499 & ";
9  $cmd .= "php " . APPLICATION_PATH . "/scripts/Calculations.php 37485 2499 ; ";
10
11 exec($cmd);
```

Snippet 46 - Chamada ao script Calculations várias vezes com diferentes parâmetros

Cada linha da variável $cmd representa uma linha da *Shell* do *Debian* que executa um ficheiro PHP. Os "&" no fim de cada linha permitem que cada um destes *scripts* corra paralelamente. Os parâmetros (os números inteiros), representam qual a gama de registos da base de dados sobre os quais cada um destes processos deve efetuar os cálculos. A última linha acaba com ponto e virgula fazendo com que o sistema fique à espera que todos os comandos anteriores sejam executados antes de prosseguir. Quando isso acontece, corre o *script Current QoS Maintenance*.

4.5.7.2 - Script Calculations

Este *script* recebe como parâmetros dois valores (identificados por $argv[1] e $argv[2], por virem da linha de comandos), instancia um objeto *Businessrules Model Calculations* (do módulo das *Business Rules*), e chama a função *Get List* com os dois argumentos recebidos, iniciando assim os cálculos de um conjunto de dados (*Snippet* 47).

```
3  $businessRules = new Businessrules_Model_Calculations;
4  $businessRules->getList($argv[1], $argv[2]);
```

Snippet 47 - Criação de um novo objeto e chamada à função Get List

4.5.7.3 - Script Current QoS Maintenance

Este *script* é responsável por normalizar os valores de QoS e tráfego contabilizado dos *MAC Addresses* que têm mais que uma entrada na tabela *Current QoS*. Isto quer dizer que em vez de termos um *MAC Address* com dois IPs diferentes, um com QoS 6, atualizado há uma semana, e o outro com QoS 7, atualizado há 2 minutos, temos ambos com QoS 7. Para fazer esta operação, primeiro o *script* verifica quais os *MAC Addresses* que têm mais que uma entrada na tabela (*Snippet* 48).

```
3  $sqlList = "SELECT macAddress, COUNT(*) c "
4           . "FROM accounting.currentQos "
5           . "WHERE macAddress IS NOT NULL "
6           . "GROUP BY macAddress HAVING c > 1";
7
8  $list = $db->fetchAll($sqlList);
```

Snippet 48 - Listar os MAC Address com mais de um registo na tabela

De seguida, o sistema percorre a lista gerada e para cada *MAC Address* busca o seu QoS e tráfego contabilizado onde o campo da data contém o valor mais recente. De seguida, é só atualizar todos os registos na tabela para aquele *MAC Address* com o QoS e tráfego contabilizado obtidos (*Snippet* 49).

```
3  foreach($list as $row => $macAddress) {
4
5      $sql = "SELECT accountedTraffic, sceQos, internalQos FROM currentQos "
6           . "WHERE macAddress = '$macAddress[macAddress]' "
7           . "AND dateTime = (SELECT max(dateTime) "
8           . "FROM currentQos WHERE macAddress = '$macAddress[macAddress]')";
9
10     $data = $db->fetchRow($sql);
11     $db->update('currentQos', $data, "macAddress = '$macAddress[macAddress]'");
12
13 }
```

Snippet 49 - Atualizando o QoS e tráfego contabilizado dos MAC Address repetidos

4.5.7.4 - Script Send Data

Este *script* corre de 15 em 15 minutos e é responsável por enviar os dados da tabela *Current QoS* para o sistema *Policy Server*. Para fazer isto, instancia um novo cliente SOAP que recebe como parâmetro o WSDL disponibilizado pelo *Policy Server* ao *Accounting System* e envia os dados como objeto (*Snippet* 50). Este script implementa o requisito funcional 4.

```
3  $client = new Zend_Soap_Client("http://policyserver.nosmadeira.pt/accounting/index/wsdl");
4  $array = $db->fetchAll("SELECT subnet AS ip, sceQos AS package
5                          FROM currentQos
6                          WHERE sceQos IS NOT NULL", null, Zend_DB::FETCH_OBJ);
7
8  $client->pushBulk($array);
```

Snippet 50 - Script Send Data

4.5.7.5 - Script Hourly Traffic

Este *script*, que corre aos 5 minutos de todas as horas através do *cron*, agrupa o tráfego das últimas horas feito por cada *MAC Address*, a partir da tabela *Min Traffic*, e insere-o como um único registo por *MAC Address* na tabela *Hourly Traffic*. Inicialmente, este *script* só ia buscar o tráfego da última hora, mas devido à possibilidade de poder falhar, haver problemas com o sistema, ou outra situação não especificada, passou a ir buscar o tráfego das últimas 24h. Desde que o sistema não fique em baixo ou a dar problemas durante 24h seguidas, não existem implicações para as regras de negócio. A cláusula IGNORE serve para quando os dados já existem, a *query* continuar a executar sem dar erro, não reinserindo os dados. O código está no *Snippet* 51.

```
for($i = 1; $i<=24; $i++) {
    $today = date('Y-m-d H', strtotime("-$i hour"));
    $hour = date('H',strtotime("-$i hour"));

    $sql = "INSERT IGNORE INTO hourlyTraffic
            SELECT macAddress, SUM(accountedTraffic), '$today' FROM minTraffic
            WHERE DATE(dateTime) = DATE('$today')
            AND HOUR(dateTime) = '$hour' GROUP BY macAddress";

    $result = $db->getConnection()->exec($sql);
}
```

Snippet 51 - Script Hourly Traffic

4.5.7.6 - Script Daily Traffic

Este *script* é em tudo semelhante ao *Hourly Traffic,* mas insere os dados na tabela *Daily Traffic*, após os agrupar a partir da tabela *Hourly Traffic*, e vai buscar o tráfego dos últimos 7 dias. É executado à meia-noite de todos os dias, via *cron.* O código está no *Snippet* 52.

```
for($i = 1; $i<=7; $i++) {
    $yesterday = date('Y-m-d', strtotime( "-$i days" ));

    $sql = "INSERT IGNORE INTO dailyTraffic
            SELECT macAddress, SUM(accountedTraffic), '$yesterday'
            FROM hourlyTraffic WHERE DATE(dateTime) = '$yesterday'
            GROUP BY macAddress";

    $db->getConnection()->exec($sql);
}
```

Snippet 52 - Script Daily Traffic

4.5.7.7 - Script Delete Traffic

Este *script*, que corre através do *cron* do *Debian* de minuto a minuto, verifica a existência de dados em lista de espera para apagar, ou seja, na tabela *Delete Traffic*, onde o campo *Deleted* esteja a 0. Se encontrar algum registo que satisfaça a condição, apaga os dados nas tabelas de acordo com o campo *All Data*, onde 0 significa que é para apagar apenas das tabelas *Min Traffic*, *Hourly Traffic* e *Daily Traffic* e 1 significa que é para apagar todos os registos relativos ao *MAC Address* especificado. Este *script* tem um *lock* no início para que não possa correr mais que uma instância do mesmo paralelamente.

4.5.7.8 - Script Partitions

Este *script*, que corre de hora a hora no *cron* do *Debian*, é responsável por criar partições para o dia seguinte para cada uma das tabelas de tráfego – *Traffic*, *QoS History*, *Min Traffic*, *Hourly Traffic* e *Daily Traffic*. Também é responsável por apagar as partições que já não são necessárias em cada uma das tabelas referidas. Nas tabelas *Traffic*, *QoS History* e *Daily Traffic* são 31 dias, na tabela *Hourly Traffic* são 7 dias e na *Min Traffic* é um dia. No entanto,

acrescentamos dois dias de vida às partições de cada uma das tabelas, por uma questão de segurança caso seja necessário algum dado e visto que não afeta o desempenho do sistema. No *Snippet* 53 podemos ver o *script* a tratar das tabelas que têm registos até 30 dias (as outras duas tabelas são feitas à parte mas o código é semelhante):

```
3  $tables = array('traffic', 'dailyTraffic', 'qosHistory');
4
5  foreach ($tables as $table) {
6
7      for ($i=33; $i<=36; $i++) {
8          try {
9                  $partition = 'p' . str_replace('-', '', date('Y-m-d', strtotime("-$i day")));
10                 $sql = "ALTER TABLE $table DROP PARTITION $partition";
11                 $db->getConnection()->exec($sql);
12         } catch(Exception $e) {
13                 //Do nothing - error handling for partition deletion since if one doesn't exist
14                 //mysql launches an error
15         }
16     }
17
18     try {
19         $partition = 'p' . str_replace('-', '', date('Y-m-d', strtotime("+1 day")));
20         $partitionRange = str_replace('-', '', date('Y-m-d', strtotime("+2 day")));
21         $sql = "ALTER TABLE $table
22                 ADD PARTITION (PARTITION $partition VALUES LESS THAN (TO_DAYS($partitionRange)))";
23         $db->getConnection()->exec($sql);
24     } catch (Exception $e) {
25         //Do nothing - if partition already exists there is no need to issue an error
26     }
27 }
```

Snippet 53 - Script Partitions para criar partições e apagar antigas

4.5.8 – Interface gráfica

À medida que o projeto se foi materializando, foi construída uma interface gráfica, bastante simples, apenas para testar a inserção de regras de negócio na base de dados, e a sua associação a determinados *Access Types*. Mais tarde, foi também construída uma interface para adicionar *overrides* por sub-rede e por *MAC Address*.

É de notar que esta interface, embora presente no requisito funcional 12, foi feita da forma mais simples possível, apenas recorrendo a PHP e HTML (*HyperText Markup Language*), pois durante o processo de desenvolvimento a NOS Madeira esclareceu que apenas serviria para testes, sendo que na fase final do projeto, as ações efetuadas na interface seriam realizadas através de serviços web disponibilizados ao OSS.

4.6 – Conclusão

A implementação é a parte mais importante de um projeto. Embora a modelação permita definir o que vai ser feito e como vai ser feito, não cria o sistema. Uma implementação sem modelação é possível (embora não aconselhável e provavelmente com muitos problemas no percurso). Já uma modelação sem implementação de nada serve.

Neste Capítulo começou-se por abordar as tecnologias que foram usadas no desenvolvimento do projeto. Sem as ferramentas corretas, é impossível construir um produto de qualidade.

De seguida, foi explicada a estrutura da base de dados do sistema, facilitando a secção seguinte, onde foram explicados os módulos do subsistema *Accounting System*, descrevendo-os e à sua funcionalidade pormenorizadamente. Justificaram-se todas as soluções implementadas, numa tentativa de permitir ao leitor compreender a linha de pensamento do desenvolvedor.

Embora neste capítulo apenas se apresentem as soluções finais, dando a ideia que a implementação foi simples, esse não foi o caso. É por isso que no próximo capítulo serão apresentados os problemas mais complicados que tiveram que ser resolvidos durante o projeto.

Capítulo 5 – Testes e Resultados

Ao longo da elaboração do projeto foram encontradas diversas dificuldades. Algumas foram resolvidas de forma relativamente simples, enquanto outras tornaram-se em desafios que necessitaram uma implementação completamente diferente. Neste capítulo serão abordados os problemas mais complexos e aos quais foi dispensada uma maior quantidade de tempo. De seguida, serão apresentados os tempos de execução dos três principais módulos do sistema e, por fim, será demonstrado que todos os requisitos funcionais e não funcionais foram atingidos.

Os testes em si foram feitos ao longo de todo o processo de desenvolvimento. Usou-se a prática de *Continuous Delivery* [30], que garante que o código pode ser rapidamente e facilmente colocado num ambiente de produção. O desenvolvimento era feito numa máquina de *staging*, ou máquina de testes, onde após garantia que o código funcionava corretamente, este era colocado na máquina de produção. É de notar que até o sistema estar completo, a máquina de produção não estava realmente "em produção". Embora fizesse o processamento completo como viria a fazer na versão final, não efetuava o último passo, que era enviar os dados de QoS para o SCE da CISCO. Neste caso, era o sistema antigo que continuava a interagir com o SCE e a rede da NOS Madeira. Após a máquina entrar em produção, os papéis inverteram-se, embora o sistema antigo continuasse a receber e processar dados, para comparar com o novo. Os resultados de QoS eram iguais, demonstrando que os cálculos eram efetuados correctamente.

5.1 – Problemas na implementação e de desempenho

Nesta secção serão descritos os problemas mais pertinentes e complexos que surgiram durante o desenvolvimento do projeto e quais os passos tomados para os resolver. O mesmo será feito para problemas de desempenho, demonstrando quais as otimizações implementadas.

5.1.1 - Envio de dados do *IP Mapping* para *Accounting System*

O envio de dados do *IP Mapping* para o *Accounting System* foi, durante bastante tempo, um processo lento, devido a uma série de fatores. Sendo esta a ação principal que permite ao *Accounting System* realizar o seu trabalho, recebendo os dados que vai processar, tornou-se fulcral encontrar uma forma de otimizar este processo.

Esta lentidão era notória quando havia qualquer falha ou pausa no envio de dados, o que fazia o *IP Mapping* acumular vários lotes de dados e depois enviá-los todos de uma vez. Quando não havia dados atrasados, o *IP Mapping* enviava cerca de 40 000 dados por ciclo. Demorava, em média, 60 segundos desde o início do envio dos dados até estarem todos guardados na base de dados, mas como o *Accounting System* conseguia processar os 40 000 dados em menos que 15 minutos, que é o intervalo de tempo entre o envio de dados, não era problema. No entanto, ao receber centenas de milhares de dados e, consequentemente, não

conseguir processá-los todos num só ciclo, o MySQL do *Accounting System* ficava sobrecarregado quando subitamente tinha que inserir, atualizar e apagar uma grande quantidade de dados ao mesmo tempo, na mesma tabela.

O envio dos dados demorava 120 segundos, em média, desde que o *IP Mapping* chamava o serviço web até ao momento em que obtinha a resposta (o que significava que os dados tinham sido todos guardados na base de dados). Esta demora gerava *timeouts*. Inicialmente, julgou-se que o problema estava no código do serviço web. Os dados eram inseridos um a um, ou seja, o *array* recebido do *IP Mapping* era percorrido em PHP e, para cada elemento, era feito um INSERT na base de dados. Os dados eram inseridos com recurso a transações na base de dados, para garantir que a operação era atómica. Caso algum elemento não fosse inserido corretamente, a função retornava imediatamente uma mensagem de erro e escrevia a causa num ficheiro de log. Era necessário esperar que o serviço fosse chamado novamente para voltar a tentar inserir todos os dados. Além da lentidão, era criada demasiada carga no servidor MySQL devido às várias conexões criadas com cada INSERT (cerca de 40 000).

Decidiu-se, então, implementar *queries* preparadas. Uma *query* preparada consiste na criação de um comando SQL no servidor MySQL sem parâmetros. De forma a compreender melhor, tomemos o exemplo do *Snippet* 54:

```
3  $sql = 'INSERT INTO traffic (macAddress, ip) VALUES (?,?)';
4  $stmt = new Zend_Db_Statement_Mysqli($db, $sql);
5  $stmt->execute(array('89365ab5c90a', '213.128.8.8'));
```

Snippet 54 - Query preparada

O comando de INSERT é criado através da variável $sql, mas na secção VALUES temos dois pontos de interrogação, que funcionam como *placeholders*. Logo de seguida, enviamos o comando para o servidor MySQL, ficando este à espera de o executar com valores reais. Finalmente, na linha 5, executamos a *query* atribuindo os valores necessários, neste caso um *MAC Address* e um IP. Ao executar a *query* num ciclo *for* e percorrendo o *array*, o MySQL está a receber apenas os valores específicos a inserir, neste caso o *MAC Address* e o IP, em vez de receber a estrutura completa da *query*. Isto torna-se mais eficiente do que criar um comando INSERT completamente novo para cada registo. Desta forma, os tempos de execução melhoraram, demorando em média 80 segundos. Era uma melhoria significativa, mas ainda não era suficiente.

A solução final que ficou implementada foi construir um único INSERT numa só *string*, com todos os valores lá. Esta solução já foi abordada na secção 4.3.1. Mais uma vez, a melhoria foi significativa – todo o processo de envio de dados passou de 80 para 50 segundos, em média.

No entanto, continuava a demorar demasiado. Ao ver as conexões à base de dados com o comando "SHOW PROCESSLIST", verificou-se que, quase instantaneamente após o *IP Mapping* chamar o serviço web, o MySQL estava a inserir os dados. Logo, o problema não era no PHP nem na criação da *query.* Foi feita uma verificação à tabela e descobriu-se que, no início do projeto, tinham sido criados determinados índices na mesma, julgando-se que seriam necessários no futuro. De cada vez que se inserem dados na tabela, esses índices são atualizados, tornando o processo de inserção mais lento. Como o sistema já estava numa fase mais avançada de desenvolvimento, foi possível reavaliar a necessidade de cada um dos

índices com mais precisão e apagar os desnecessários. Esta reavaliação também foi feita em todas as outras tabelas da base de dados. Os resultados foram claros – por cada 40 000 dados, o sistema passou de 50 segundos para 5 segundos para inserir todos os dados, ou seja, uma melhoria de 90%.

5.1.2 - Regra de negócio *Time Weight*

Aquela que é a regra de negócio mais importante foi também a mais complexa de implementar, não pela lógica em concreto mas pelo seu desempenho, que inicialmente se revelou bastante fraco. Como foi visto na secção 4.5.5, esta regra soma o tráfego entre dois períodos de tempo e aplica uma percentagem/peso ao valor obtido. A primeira implementação consistia em ir buscar todo esse tráfego à tabela *Traffic*. Quando o sistema começou a receber e processar dados, para ir testando o funcionamento e comparando os resultados com os resultados do sistema anterior, a tabela estava vazia e, como tal, não havia qualquer problema. A regra era executada rapidamente. No entanto, à medida que a tabela foi enchendo, o sistema começou a ficar cada vez mais lento, até ao ponto de demorar em média 94,5 segundos para processar cada 1000 registos. É de notar que estes testes foram feitos na máquina de *staging*, que tinha capacidade de desempenho muito inferior à de produção. O código era semelhante ao que foi implementado na versão final, mas em vez de somar o tráfego das três tabelas resumo, somava apenas o que estava na tabela *Traffic*, como se pode ver no *Snippet* 55:

```
3  "SELECT macAddress, SUM(accountedTraffic) AS accountedTraffic FROM traffic
4   WHERE macAddress = '$traffic[macAddress]'
5   AND dateTime BETWEEN DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[endsOn])
6   AND DATE_SUB('$traffic[dateTime]', INTERVAL $businessRule[startsOn])";
```

Snippet 55 - Código da regra de negócio Time Weight antes das alterações

Quando se detetaram os problemas de desempenho, foram propostas três soluções, durante um *brainstorming* com a NOS Madeira. A terceira e última foi a solução final, já descrita na secção 4.5.5. Aqui descrevem-se as duas soluções prévias que foram sugeridas. Todas as soluções consistiam em criar tabelas de resumo do tráfego, diferindo apenas na forma como esses dados de resumo seriam criados e quando.

A primeira solução consistia em criar uma tabela que tivesse 4 campos – *MAC Address, timestamp*, "a partir de" e "até". Os primeiros dois campos não necessitam de explicação. O campo "a partir de" indicaria o tráfego feito pelo respetivo *MAC Address* naquela data/hora/minuto até às 23h:59m daquele dia. O campo "até" fazia o inverso, indicando para o respetivo *MAC Address* naquela data/hora/minuto qual o tráfego que tinha sido feito. Depois, a regra de negócio apenas tinha que somar os dados de alguns campos. O tráfego total de cada dia podia ser obtido somando os valores dos campos "a partir de" e "até" de qualquer registo daquele dia. Sempre que não fosse necessário contabilizar o tráfego todo, bastava usar um dos dois campos. No entanto, ainda durante o *brainstorming* com a NOS Madeira, foi decidido abandonar esta ideia ao surgir outra de implementação mais simples, cuja explicação se segue.

A segunda solução proposta foi semelhante à que foi implementada, com as três tabelas de resumo, mas com uma diferença – não haveria tráfego repetido. Ou seja, qualquer tráfego que fosse apagado da tabela *Min Traffic* iria automaticamente para a *Hourly Traffic* e qualquer tráfego que fosse apagado da tabela *Hourly Traffic* iria automaticamente para a tabela *Daily Traffic*. A *query* da regra de negócio seria praticamente igual à que foi implementada, retirando apenas a cláusula que indica que para cada tabela só se podem usar os registos a partir de uma determinada data. Desde que o tráfego não fosse repetido, essa cláusula revelava-se inútil. No entanto, para simplificar o processo de criação de registos de tráfego para as tabelas de resumo, decidiu-se deixar tráfego repetido, incluindo a dita cláusula na regra de negócio. A diferença de desempenho foi extremamente elevada: para processar 1000 registos, passou-se de uma média de 94,5 segundos para 11,8 segundos. Assumindo um lote normal de 40 000 dados por ciclo, o tempo de processamento total desceu de 63 minutos para 8 minutos, ou seja, uma melhoria em 87,5%, como se pode ver na Figura 14:

	Média para 1000 registos	
	Tabela Traffic	**Tabelas resumo**
	75	18
	63	14
	79	15
	58	8
	76	8
	73	8
	84	8
	191	8
	148	8
	117	13
	78	24
	81	16
	61	10
	116	10
	61	11
	187	11
	76	11
	77	11
TOTAL	**94,5**	**11,8**

Figura 15 - Diferença de tempos com tabelas resumo

5.1.3 - Scripts Delete Old Traffic e Partitions

Inicialmente, foi criado um *script* que corria antes de cada ciclo de processamento, que apagava todo o tráfego da tabela *Min Traffic* com mais de 24h, da tabela *Hourly Traffic* com mais de 7 dias e das tabelas *Daily Traffic, Traffic* e *QoS History* com mais de 30 dias. Ou seja, a cada 15 minutos apagava cerca de 40 000 registos da tabela de *Min Traffic*, a cada hora apagava outros 40 000 registos da tabela de *Hourly Traffic* e a cada 24h apagava outros 40 000 registos das tabelas *Daily Traffic, Traffic* e *QoS History*. No entanto, isto trouxe problemas no processamento. Se por alguma razão o sistema tivesse que ser pausado, o número de registos para apagar acumulava, e ao correr o *script* este atrasava o sistema todo, criando *locks* nas

tabelas da base de dados e impedindo outros processos de serem executados corretamente, acabando sempre por dar erro por *Dead Lock*. O código do *script* está no *Snippet* 56:

```
3  $sqlMinute  = "DELETE FROM minTraffic WHERE dateTime < DATE_SUB(NOW(), INTERVAL 1440 MINUTE)";
4  $sqlHourly  = "DELETE FROM hourlyTraffic WHERE dateTime < DATE_SUB(NOW(), INTERVAL 10080 MINUTE)";
5  $sqlDaily   = "DELETE FROM dailyTraffic WHERE dateTime < DATE_SUB(NOW(), INTERVAL 30 DAY)";
6  $sqlTraffic = "DELETE FROM traffic WHERE dateTime < DATE_SUB(NOW(), INTERVAL 30 DAY)";
7  $sqlHistory = "DELETE FROM qosHistory WHERE dateTime < DATE_SUB(NOW(), INTERVAL 30 DAY)";
8
9  try {
10     $db->getConnection()->exec($sqlMinute);
11     $db->getConnection()->exec($sqlHourly);
12     $db->getConnection()->exec($sqlDaily);
13     $db->getConnection()->exec($sqlTraffic);
14     $db->getConnection()->exec($sqlHistory);
15 } catch (Exception $e) { /* */ }
```

Snippet 56 - Script DeleteOldTraffic

Foi, então, que se decidiu alterar as partições. A ideia inicial (e que acabou por ser a implementada) era criar partições por data e hora. No entanto, inicialmente as partições foram criadas por *MAC Address*. Esta decisão foi tomada porque os cálculos são feitos por *MAC Address*. Os registos das tabelas são agrupados por *MAC Address*. A grande maioria das *queries* tem o *MAC Address* na cláusula "*WHERE*". Foram criadas 83 partições, por "*range*" ou gama, de acordo com sintaxe exemplificada no *Snippet* 57:

```
3  ALTER TABLE traffic
4      PARTITION BY RANGE COLUMNS(macAddress) (
5      PARTITION p1 VALUES LESS THAN ('0005ca6'),
6      PARTITION p2 VALUES LESS THAN ('0005ca7'),
7      PARTITION p3 VALUES LESS THAN ('0005ca8'),
8      (...);
9      PARTITION p82 VALUES LESS THAN ('e448c7c'),
10     PARTITION p83 VALUES LESS THAN MAXVALUE
11 )
```

Snippet 57 - Partições por MAC Address

Como as partições funcionam como sub-tabelas, ao procurar registos por *MAC Address* o sistema procurava numa tabela mais pequena, melhorando o desempenho. No entanto, essa melhoria era pouco significativa. Por outro lado, também era inútil apagar tráfego antigo de 15 em 15 minutos, desde que as regras de negócio que usam o tráfego (principalmente a *Time Weight*) tivessem os seus períodos de tempo bem definidos. Com isto em conta, tomou-se a decisão de alterar as partições e voltar à ideia inicial de as criar por dia, como foi visto na secção 4.2.12. A melhoria foi instantânea. Fazer um DROP PARTITION é instantâneo e criar uma nova partição (se ainda não existirem dados para inserir na mesma) é também instantâneo. Os *Dead Locks* que apareciam devido aos DELETES individuais desapareceram completamente.

5.1.4 - Serviços Web para apagar dados

À semelhança do problema do *script Delete Old Traffic*, os serviços web *Delete Traffic History* e *Delete History* davam problemas ao serem executados. Mas, ao contrário do *script*, o problema era um pouco diferente. Ao chamar qualquer um destes serviços web que, como foi visto nas secções 4.5.6.12 e 4.5.6.13, recebem um *MAC Address*, o serviço chamado começava automaticamente a apagar os dados (*Snippet* 58):

```
3  $sqlMinute  = "DELETE FROM minTraffic WHERE macAddress = '$macAddress'";
4  $sqlHourly  = "DELETE FROM hourlyTraffic WHERE macAddress = '$macAddress'";
5  $sqlDaily   = "DELETE FROM dailyTraffic WHERE macAddress = '$macAddress'";
6  $sqlTraffic = "DELETE FROM traffic WHERE macAddress = '$macAddress'";
7  $sqlHistory = "DELETE FROM qosHistory WHERE macAddress = '$macAddress'";
8
9  try {
10     $db->getConnection()->exec($sqlMinute);
11     $db->getConnection()->exec($sqlHourly);
12     $db->getConnection()->exec($sqlDaily);
13     $db->getConnection()->exec($sqlTraffic);
14     $db->getConnection()->exec($sqlHistory);
15 } catch (Exception $e) { /* */ }
```

Snippet 58 - Código antigo do serviço Delete Traffic History

O problema estava na demora em responder ao cliente do serviço web. Os DELETES demoravam um tempo considerável e o cliente obtinha um *timeout*, sem saber se os dados tinham sido apagados ou não. Por sua vez, chamar o serviço mais que uma vez ao mesmo tempo criava vários DELETES ao mesmo tempo que, por vezes, geravam *dead locks* noutros processos do sistema, como por exemplo nos cálculos. Devido a esta situação, foi implementada a solução da fila de espera já vista na secção 4.5.6.12, eliminando grande parte dos problemas e sendo muito mais eficiente. Embora ainda exista a possibilidade de *dead locks*, esta foi reduzida por não existirem vários processos de apagar dados a correr paralelamente.

5.1.5 - Geração de dados para a tabela *Min Traffic*

Uma das questões iniciais de quando se avançou com a ideia das três tabelas de resumo de tráfego foi: a tabela *Daily Traffic* vai buscar dados à *Hourly Traffic*, a *Hourly Traffic* vai buscar dados à *Min Traffic*, mas onde vai buscar dados a tabela *Min Traffic*? Durante algum tempo, a implementação consistia em chamar um *script* assim que o *IP Mapping* acabava de enviar os dados, que buscava os dados da tabela *Traffic* onde o campo *Is Accountable* estivesse a NULL, como se pode ver no *Snippet* 59:

```
3  $sql = "INSERT IGNORE INTO minTraffic
4          SELECT idTraffic, macAddress, accountedTraffic, dateTime
5          FROM traffic WHERE isAccountable IS NULL";
```

Snippet 59 - Script Min Traffic

Embora não desse para inserir dados repetidos devido à chave primária *Id Traffic*, havia a possibilidade de "perder" dados caso o ciclo de processamento arrancasse ao mesmo

tempo, ainda que tal situação fosse altamente improvável. Foi aí que se implementou a solução já vista na secção 4.3.1, de inserir os dados na tabela *Min Traffic* já no serviço web disponibilizado ao *IP Mapping*. O serviço web passou a demorar em média 10 segundos a executar em vez de 5, mas a solução ficou mais elegante e a diferença de tempo foi desprezável, visto que este serviço era chamado, por norma, apenas uma vez a cada 15 minutos.

5.1.6 - Falta de espaço em disco

Um dos problemas iniciais do projeto, assim que se começaram a gerar grandes quantidades de dados, foi a falta de espaço em disco, mesmo apagando dados antigos constantemente. Após averiguação, descobriu-se que o problema estava no fato de os DELETES não reclamarem o espaço em disco de volta. Mesmo fazendo TRUNCATE (que apaga todos os dados de uma tabela) ou DROP (que apaga completamente a tabela), o ficheiro "*ibdata*", que contém os dados todos da base de dados mantém o mesmo tamanho, apenas realocando o espaço se houver novos dados para substituir os antigos, ou então ocupando mais espaço quando necessário. Rapidamente se chegou à conclusão que, com o tempo, este ficheiro iria crescer até ao limite da capacidade de armazenamento do disco. A solução foi ativar a variável "*innodb_file_per_table*" [31] que cria um ficheiro .ibd para cada tabela (e como algumas tabelas têm partições, um ficheiro .ibd para cada partição). Desta forma, um DROP TABLE (ou DROP PARTITION) recupera o espaço em disco.

5.1.7 - Colocação de dados em memória

Na função *Get Business Rules* da classe *Calculations* foi feito um teste para verificar se era mais rápido aceder ao mapeamento *Access Type* – Regra de negócio em SQL para cada registo, ou colocar todos os mapeamentos em memória e aceder-lhes com o PHP. Um teste semelhante foi feito ao mapeamento de *Access Type* – tráfego/prioridade. A melhoria no desempenho depois de colocar estes dados em memória foi notória, quando ainda na máquina de testes foi feita uma comparação entre este método ou uma consulta à base de dados de cada vez que era necessário aceder aos mesmos. A comparação pode ser verificada na Figura 15, que mostra, em segundos, o tempo completo de processamento de 1000 registos:

	AcessTypeToBusinessRules		AccessTypePriority	
	SQL	Memória	SQL	Memória
	14,2	16,7	16,7	14,1
	14,2	14,1	14,1	14
	14	13,8	13,8	13,8
	14,2	14,2	14,2	13,8
	18	14,2	14,2	13,8
	16,8	13,7	13,7	13,7
	14,7	14,2	14,2	14
	14,7	13,9	13,9	13,8
	14,4	13,8	13,8	14,3
	14,3	13,8	13,8	13,5
	14,4	13,7	13,7	13,6
	14,6	14	14	13,7
	17,7	13,8	13,8	13,2
	14,8	13,6	13,6	13,5
Média:	**15,07**	**14,11**	**14,11**	**13,77**

Figura 16 - Diferenças de tempo, em segundos, entre dados em memória e SQL

Embora a máquina de testes tivesse um desempenho significativamente mais fraco que a máquina de produção, com estas alterações a diferença total dá menos 1,3s por cada 1000 registos, o que para 40000 registos dá cerca de 52 segundos de diferença no processamento total, uma redução em 8,6%.

5.2 – Tempos de execução

Após colocar o sistema em produção, foram feitos testes ao tempo que cada módulo demorava a executar as suas principais funções: recebimento de dados no módulo *IP Mapping*, cálculos no módulo *Accounting System* e envio de dados no módulo *Policy Server*. Os resultados, em segundos, foram obtidos num ambiente de execução real, sendo por isso valores corretos e que representam com precisão o desempenho do sistema. Assumindo que cada fase do ciclo de processamento ocorre imediatamente a seguir à anterior, ou seja, o sistema começa a processar os dados assim que os recebe do *IP Mapping*, e envia-os ao *Policy Server* assim que acaba os cálculos, é possível fazer um ciclo completo de processamento em 99,6 segundos (14,4 + 83,3 + 1,9). Os valores podem ser vistos na Figura 17. Estes tempos estão dentro do intervalo de tempo estabelecido no requisito não funcional 4, em que o sistema tem que efetuar um ciclo completo de processamento num intervalo de 15 minutos.

	IPMapping	Cálculos	Policy Server
	18,7	101,4	1,8
	10,1	106,6	1,8
	10,7	66,3	1,9
	10,7	66,9	1,9
	11,9	70,0	1,9
	11,9	72,2	1,9
	12,1	73,3	1,9
	13,3	74,3	1,9
	13,4	79,3	1,9
	15,5	80,5	1,9
	15,9	82,3	1,9
	16,2	82,7	2,0
	17,8	83,8	2,0
	18,4	84,8	2,0
	20,9	85,3	2,0
	21,4	88,6	2,0
	27,1	89,0	2,0
	6,5	91,9	2,0
	7,8	92,8	2,1
	8,4	93,6	2,2
Média:	**14,4**	**83,3**	**1,9**

Figura 17- Tempos de execução dos 3 módulos do subsistema Accounting System

5.3 – Requisitos cumpridos

Como foi visto ao longo do Capítulo 4, a grande maioria dos requisitos funcionais foram cumpridos. A exceção foi o requisito 11. Em vez de implementar todas as funcionalidades do sistema como serviços, o que faria com que fosse necessário permitir, por exemplo, iniciar o processo de cálculos via serviço web, apenas foram implementados os serviços que a NOS Madeira achou serem necessários.

Quanto aos requisitos não funcionais, todos foram cumpridos. Os requisitos não funcionais 1, 2, 3 e 5 podem ser comprovados no Capítulo 4.

O requisito não funcional 4 pode ser comprovado na secção 5.2.

O requisito não funcional 6 foi cumprido pois, desde a data de entrada do projeto em produção, a 18 de Agosto de 2014, até à data de finalização do presente documento, a 23 de Outubro de 2014 (um período de 66 dias), o sistema apenas parou quando foram feitas alterações importantes no mesmo, intencionalmente, criando algum *downtime*.

Não foi possível testar cuidadosamente o requisito não funcional 7. No entanto, podemos assumir que, visto que o sistema consegue efetuar um ciclo completo de processamento em 99,6 segundos, consegue recuperar 20 ciclos perdidos no espaço de uma hora, pois 20 ciclos a 99,6 segundos cada, demoram 1992 segundos, bem dentro do espaço dos 3600 segundos que constituem uma hora.

Finalmente, o requisito funcional 8 foi cumprido, como explicado na secção 3.5.4.

5.4 – Conclusão

A complexidade de um projeto de Engenharia de *Software* é, por vezes, subestimada ou incompreensível, exceto para aqueles que têm que garantir a sua qualidade sem perder de vista o objetivo final. Todas as dificuldades, problemas, obstáculos e soluções encontradas são importantes, pois estas adicionam valor a qualquer projeto, demonstrando que a sua execução não foi um processo simples e direto, sem obstáculos no caminho.

Neste capítulo foram apresentados diversos problemas, alguns de cariz mais simples e outros mais difíceis, mas todos eles pertinentes o suficiente para diminuir o dito valor do projeto, caso não tivessem sido resolvidos.

De seguida, apresentaram-se tempos de execução do processamento do sistema, importantes para perceber como, apesar de toda a complexidade, o desempenho não foi deixado de parte.

Para terminar, no próximo capítulo serão apresentadas as conclusões retiradas do projeto.

Capítulo 6 – Conclusões

O desenvolvimento deste projeto foi de valor incálculavel, tanto para o cliente, como para o desenvolvedor. A NOS Madeira teve a possibilidade de manter em funcionamento uma lógica de negócio que é, desde há largos anos, um dos seus pontos fortes, e que a diferencia da competição. Recriar um sistema de raiz, mantendo a sua funcionalidade base, e expandindo-a para responder às novas necessidades da empresa, foi uma decisão que se revelou correta. Não só o novo sistema é mais robusto e fácil de compreender, como permite que sejam feitas alterações no seu funcionamento, ao contrário do sistema anterior. A longo prazo, estas caracteristicas farão a diferença e darão vantagem competitiva no mercado à NOS Madeira, permitindo que as próximas alterações feitas ao sistema não impliquem que este seja construído de raíz, mas sim expandindo-o e aproveitando o trabalho já feito.

Para o desenvolvedor, o valor do projeto foi ainda maior. Foi possível aplicar conhecimentos adquiridos ao longo do curso, mas, mais importante ainda, foi dada a possibilidade de adquirir muitos outros, estando em contato com um sistema fulcral para a garantia de qualidade de serviço da NOS Madeira.

A nível técnico, foram aprofundados conhecimentos em linguagens de programação, como o PHP, e em sistemas gestores de bases de dados relacionais, como o MySQL. A utilização de uma *framework* para desenvolver o projeto foi, também, um fator de grande valor para os conhecimentos adquiridos.

A implementação de serviços web, embora facilitada pelas ferramentas da *Zend Framework*, gerou bastante interesse e mostrou o quão importante é garantir qualidade.

Finalmente, o contacto com um sistema operativo como o Debian, da familia UNIX, apenas usando linha comandos, trouxe bastante interesse por ser uma forma de comunicar com um sistema operativo pouco habitual para o desenvolvedor.

Ter que fazer alterações ao sistema após este já estar em produção gerou duas situações distintas: uma em que o sistema era parado, sendo necessário agir rapidamente e sem qualquer margem para erro, de forma a evitar *downtime* extenso, e outra em que o sistema continuava a funcionar normalmente, sendo necessário um cuidado extremo para não causar qualquer erro que pudesse afetar o *output* final do mesmo.

Embora tais alterações fossem feitas, na maior parte das vezes, numa máquina de *staging* e não na de produção, certos detalhes, como por exemplo, o URL para o WSDL dos serviços web ser diferente em ambas as máquinas, impediam que a transição das alterações de uma máquina para a outra corresse como o esperado. Estes problemas eram muitas vezes de natureza simples, mas nem sempre fáceis de detetar rapidamente. Analisar o problema e corrigi-lo num curto espaço de tempo foi uma virtude obtida neste projeto, fruto de tais situações.

O projeto foi uma ótima oportunidade para estar em contato com uma empresa cujos serviços são usados 24h por dia, 7 dias por semana. Este fator leva à necessidade de saber estimar corretamente prazos num projeto, saber o que pode e o que não pode ser feito e conseguir cumprir todos os objetivos dentro dos prazos estipulados. Uma falha em qualquer um destes pontos pode levar a insatisfação por parte dos clientes, prejudicando o bom nome da empresa. A boa comunicação entre o desenvolvedor e a NOS Madeira foi de grande

importância, permitindo que qualquer problema fosse resolvido rapidamente e com uma equipa pronta a intervir e ajudar em caso de ser um problema extremamente critico.

O resultado final foi um sistema estável, rápido e que cumpre todos os objetivos propostos no início do projeto. É de notar que, embora o subsistema de accounting tenha tido apenas um desenvolvedor, não seria possível ser bem-sucedido sem a preciosa colaboração, disponibilidade e espirito de entreajuda dos membros da NOS Madeira, e dos desenvolvedores dos subsistemas *IP Mapping* e *Policy Server*.

6.1 – Trabalho futuro

Apesar do sistema estar correntemente em produção e a cumprir com os objetivos, há sempre margem para melhoria. Entre essas melhorias, sugere-se:

- Realização de testes intensivos aos vários componentes do sistema, para encontrar *bugs* e corrigi-los o mais rapidamente possível. Devido a constrangimentos de tempo não foi possível efetuar tais testes, tendo o sistema entrado em produção a partir do momento em que se considerou estável e com as funcionalidades implementadas;
- Standardização das respostas aos serviços web nos módulos *IP Mapping* e *Policy Server*. Estes dois módulos não obedecem aos mesmos padrões de comunicação presentes nos serviços web disponibilizados ao OSS;
- Refactoração do código. Embora este processo tenha sido feito várias vezes ao longo do processo de desenvolvimento, é sempre possível melhorar, particularmente se o código for revisto por um desenvolvedor com mais experiência;
- Fazer uma revisão à base de dados e uma normalização das tabelas. Alguns atributos poderão ser desnecessários, como alguns IDs, e sendo a base de dados um dos pontos mais criticos do sistema, afiná-la para melhorar o desempenho é um processo importante e valioso;

“Trabalho duro não garante sucesso, mas nenhum sucesso é possível sem trabalho duro.” – Dr. T.P. Chia

Referências

[1] D. Aguiar, “IP Network Usage Accounting - Parte I - IP Mapping,” Funchal, Portugal, A ser publicada.

[2] J. Canha, “IP Network Usage Accounting - Parte III - Policy Server,” Funchal, Portugal, A ser publicada.

[3] Microsoft, “Microsoft Product Lifecycle Search,” [Online]. Available: http://support2.microsoft.com/lifecycle/search/?alpha=windows+2000. [Acedido em 6 Setembro 2014].

[4] I. E. D. Center, “VBScript is no longer supported in IE11 edge mode,” [Online]. Available: http://msdn.microsoft.com/en-us/library/ie/dn384057(v=vs.85).aspx. [Acedido em 30 Setembro 2014].

[5] Microsoft, “How to use data transformation services (DTS) to export data from a Microsoft Access database to an SQL Server database,” [Online]. Available: http://support.microsoft.com/kb/285829. [Acedido em 30 Setembro 2014].

[6] Microsoft Developer Network, “OLE DB Tutorial,” [Online]. Available: http://msdn.microsoft.com/en-us/library/aa288452(v=vs.71).aspx. [Acedido em 29 Setembro 2014].

[7] Microsoft Developer Network, “Data Transformation Services (DTS),” [Online]. Available: http://msdn.microsoft.com/en-us/library/cc707786(v=sql.105).aspx. [Acedido em 11 Setembro 2014].

[8] L. Noabeb, “Anatomy of a Web Service: XML, SOAP and WSDL for Platform-independent Data Exchange,” [Online]. Available: http://www.webreference.com/authoring/web_service/index.html. [Acedido em 16 Setembro 2014].

[9] W3, “Simple Object Access Protocol (SOAP) 1.1,” 8 Maio 2000. [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Acedido em 16 Setembro 2014].

[10] J. Mueller, “Understanding SOAP and REST Basics,” 8 Janeiro 2013. [Online]. Available: http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/. [Acedido em 18 Outubro 2014].

[11] W3C, “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,” 26 Junho 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/. [Acedido em 17 Setembro 2014].

[12] D. M. Elkstein, “Learn REST: A Tutorial,” [Online]. Available: http://rest.elkstein.org/. [Acedido em 16 Setembro 2014].

[13] S. Francia, “REST Vs SOAP, The Difference Between Soap And Rest,” 15 Janeiro 2014. [Online]. Available: http://spf13.com/post/soap-vs-rest. [Acedido em 18 Outubro 2014].

[14] Z. Urlocker, “Google Runs MySQL,” 13 Dezembro 2005. [Online]. Available: http://zurlocker.typepad.com/theopenforce/2005/12/googles_use_of_.html. [Acedido em 17 Outubro 2014].

[15] J. Sobel, “Keeping Up,” 21 Dezembro 2007. [Online]. Available: https://www.facebook.com/notes/facebook/keeping-up/7899307130. [Acedido em 17 Outubro 2014].

[16] T. Haerder e A. Reuter, “Principles of Transaction-Oriented Database Recovery,” *ACM Computer Surveys,* 1983.

[17] MySQL, “MySQL :: MySQL 5.5 Reference Manual :: 14.1.2 InnoDB as the Default MySQL Storage Engine,” [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/innodb-default-se.html. [Acedido em 19 Outubro 2014].

[18] MySQL, “MySQL :: MySQL 5.5 Reference Manual :: 15.3 The MyISAM Storage Engine,” [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/myisam-storage-engine.html. [Acedido em 20 Outubro 2014].

[19] MySQL, “MySQL :: MySQL 5.5 Reference Manual :: 14.1 Introduction to InnoDB 1.1,” [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/innodb-introduction.html. [Acedido em 20 Outubro 2014].

[20] I. Sommerville, “System Modeling,” em *Software Engineering*, Addison-Wesley, 2011, pp. 119-120.

[21] M. Rouse, “Operational Support System,” Julho 2007. [Online]. Available: http://searchtelecom.techtarget.com/definition/operational-support-system. [Acedido em 14 Outubro 2014].

[22] CISCO, “Cisco SCE8000 10GBE Software Configuration Guide, Release 3.7.x,” 7 Fevereiro 2014. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/cable/serv_exch/serv_control/broadband_app/rel37x/swcfg8000_10gbe/swcfg8000_10gbe/Overview.html#pgfId-1044878. [Acedido em 3 Outubro 2014].

[23] K. Schwaber e J. Sutherland, The SCRUM guide, 2013.

[24] R. Kay, “LAMP,” 25 Setembro 2006. [Online]. Available: http://www.computerworld.com/article/2553939/app-development/lamp.html. [Acedido em 13 Setembro 2014].

[25] A. Gupta, “Scaling WikiPedia with LAMP: 7 billion page views per month,” 20 Outubro 2008. [Online]. Available: https://blogs.oracle.com/WebScale/entry/scaling_wikipedia_with_lamp_7. [Acedido em 18 Outubro 2014].

[26] S. Chacon, “Git Basics,” em *Pro Git*, 2009, pp. 4-5.

[27] B. Collins-Sussman, B. W. Fitzpatrick e C. M. Pilato, Version Control with Subversion: For

Subversion 1.7, 2011.

[28] Zend Framework, “Zend Framework & MVC Introduction,” [Online]. Available: http://framework.zend.com/manual/1.12/en/learning.quickstart.intro.html. [Acedido em 05 Setembro 2014].

[29] MySQL, “MySQL :: MySQL 5.5 Reference Manual :: 17 Replication,” [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/replication.html. [Acedido em 8 Outubro 2014].

[30] C. Caum, “Continuous Delivery Vs. Continuous Deployment: What's the Diff?,” 30 Agosto 2013. [Online]. Available: http://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff. [Acedido em 24 Outubro 2014].

[31] MySQL, “MySQL :: MySQL 5.5 Reference Manual :: 14.8.2 InnoDB File-Per-Table Mode,” [Online]. Available: http://dev.mysql.com/doc/refman/5.5/en/innodb-multiple-tablespaces.html. [Acedido em 19 Outubro 2014].

Anexos

Anexo I – Arquitetura inicial do sistema

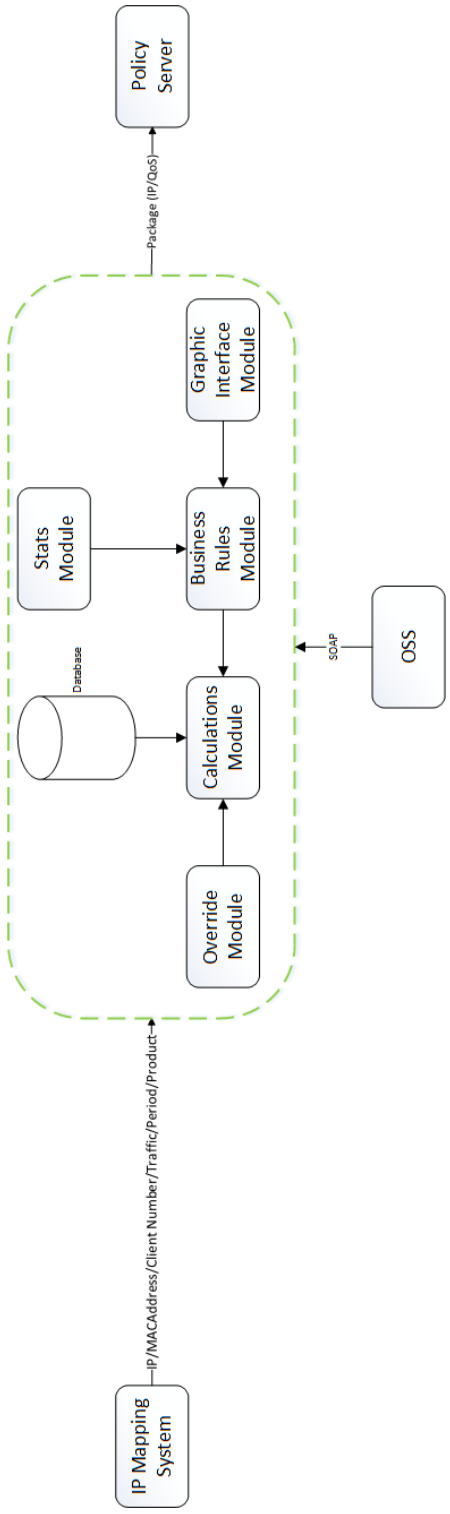

Figura 18 – Arquitetura inicial do sistema

Anexo II – Diagrama de atividades inicial

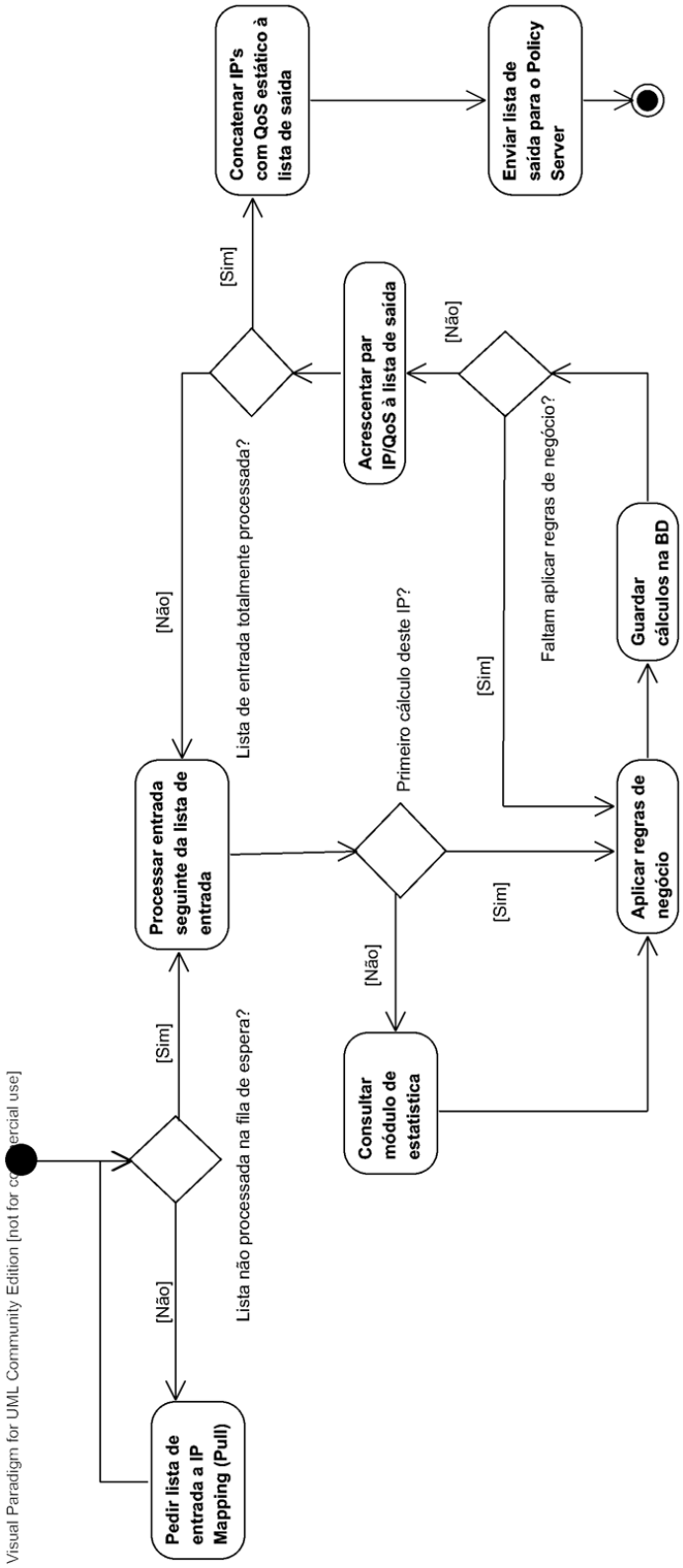

Figura 19 - Diagrama de atividades inicial

Anexo III – Diagrama de sequência

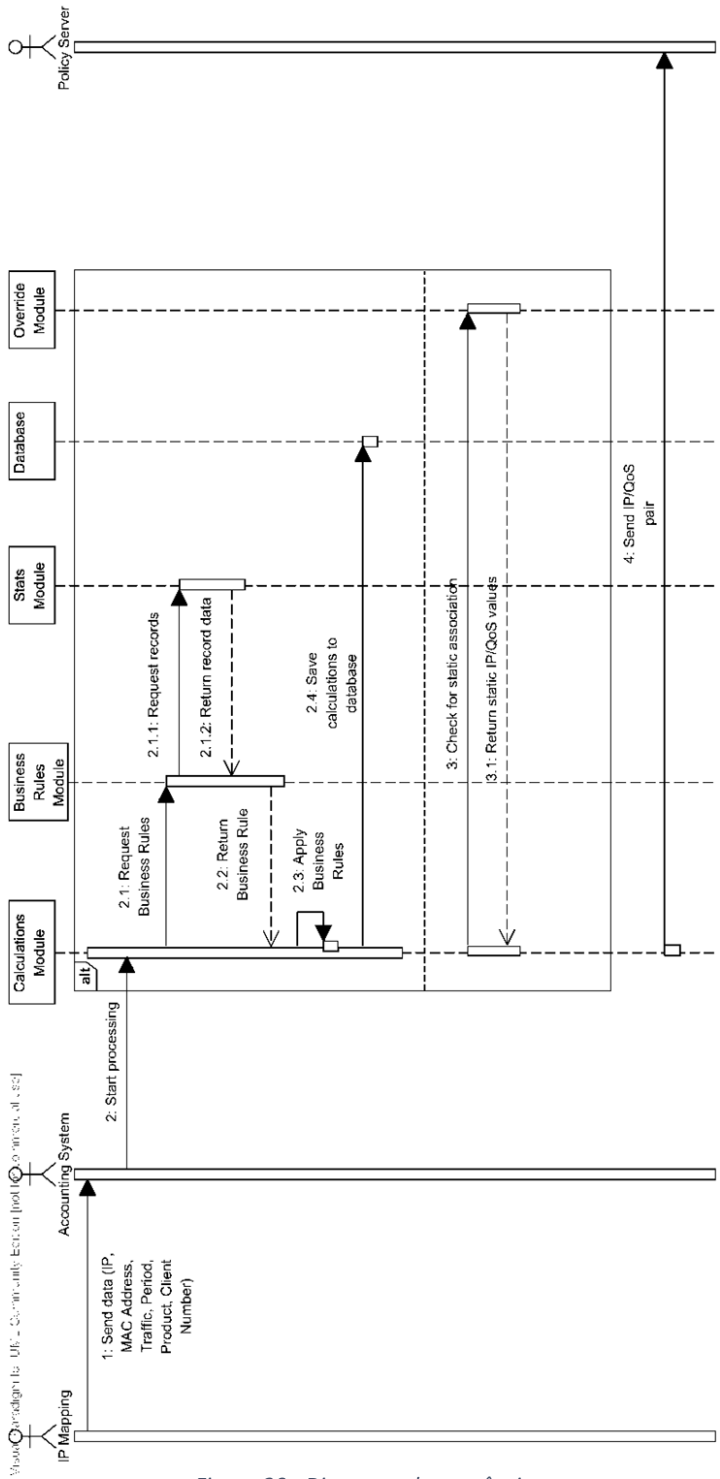

Figura 20 - Diagrama de sequência