Final

IP Network Usage Accounting - Parte III

PROJETO DE MESTRADO

Jorge Castro Freire Canha
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

outubro | 2014

M UMa
4
N IP

IP Network Usage Accounting - Parte III

PROJETO DE MESTRADO

Jorge Castro Freire Canha
MESTRADO EM ENGENHARIA INFORMÁTICA

SUPERVISOR
Lina Maria Pestana Leão de Brito
Filipe Fernandes Azevedo

CO-SUPERVISOR
Karolina Baras

Resumo

De modo a manter políticas de utilização aceitável dos seus serviços de Internet, a NOS Madeira tem usado um sistema de fabrico próprio onde os clientes são catalogados de acordo com o tráfego que realizam. Contudo, esse sistema tornou-se demasiado antigo para as necessidades atuais da empresa. Usava tecnologias descontinuadas, não tinha interfaces de integração, faltava modularidade e não tinha a flexibilidade necessária para expandir as regras de negócio.

Este projeto centra-se na implementação de um dos três subsistemas que substituem o sistema antigo: o subsistema controlador. O objetivo é modernizar, facilitar a manutenção e garantir maior flexibilidade. Tudo isto com recurso a linguagens de programação atuais como o PHP, ferramentas como a *Zend Framework* e mantendo em mente as melhores práticas de programação.

São apresentados a especificação e modelação do sistema, assim como todos os detalhes da implementação em conjunto com as decisões e problemas encontrados.

Os testes e resultados, incluindo a entrada com sucesso em produção do sistema, juntamente com sugestões de melhorias futuras concluem este trabalho.

Palavras-chave

Controlo de largura de banda

Integração

Qualidade de serviço

Serviços web

Tráfego

Abstract

In order to maintain a fair usage policies of their Internet services, NOS Madeira has used an inhouse system where customers are rated according to their bandwidth usage. However, the system grew old for the company's current needs. It was using deprecated systems and technologies, lacked integration interfaces, modularity and didn't have flexibility needed to meet the current the bussiness rules.

This project focuses on one of the three subsystems that replace the old one, the controller subsystem. It aims to modernize, make it more flexible and easier to maintain. All this is accomplished by using current programming languages like PHP, solid frameworks like the Zend Framework while keeping in mind best programming practices.

The implementation details, along with decisions and problems faced, are presented as well.

The tests, results, including a successful production deployment together with suggestions of further improvements will conclude this work.

Keywords

Bandwidth control

Integration

Quality of service

Traffic

Web services

Agradecimentos

Quero agradecer às professoras Lina Brito e Karolina Baras, na qualidade de orientadoras, pela dedicação, ajuda e conselhos dados para que este trabalho fosse realizado com sucesso.

Estendo também os meus agradecimentos a Nélio Vieira e Eng.º Filipe Avezedo da NOS Madeira. Primeiro, pela oportunidade de poder ter realizado este trabalho e, segundo, por todo o apoio prestado e simpatia. Quero igualmente agradecer à equipa do Centro de Dados da NOS Madeira pela disponibilidade e conselhos dados durante as várias fases do projeto.

Agradecer também a todos os professores da UMa que, de uma forma ou de outra, contribuíram para a minha formação com os seus ensinamentos.

E finalmente, aos meus pais e irmão, pelo apoio incondicional durante todos estes anos do meu percurso académico. Sem eles nada disto poderia ter sido possível.

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Application programming interface
BD	Base de dados
CLU	Command-Line Utilities
CMTS	Cable Modem Termination System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
INT	Integer, inteiro
IP	Internet Protocol
LEG	Login Event Generator
MVC	Model-view-controller
OSS	Operations Support System
RTMP	Real Time Messaging Protocol
SCE	Service Control Engine
SDPI	Stateful Deep Packet Inspection
SGBDR	Sistema de Gestão de Base de Dados Relacional
SM	Subscriber Manager
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access protocol
SP	Servidor de Políticas
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoIP	Voice-over-IP
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Capítulo 1

Introdução

Todos os provedores de serviços de Internet necessitam de manter um registo do tráfego realizado para dentro e para fora da sua rede, quer para fins estatísticos, quer para garantir políticas de utilização aceitável. Qualquer falha nesta gestão leva a frustração por parte dos clientes e ao congestionamento da rede.

A NOS Madeira é o principal fornecedor de Internet na Madeira com redes HFC (*Hybrid Fibre-Coaxial*), FTTH (*Fiber To The Home*) e móvel. De acordo com a Cisco [1] o tráfego global de IP aumentou 8 vezes nos anos entre 2005 e 2010, e está previsto que irá quadruplicar desde então até 2015, atingindo os 966 *exabytes*. Isto deve-se em parte à popularidade dos *smartphones* e *tablets* que fazem aumentar o consumo de tráfego. Mas também existem muitas aplicações que estão na frente deste crescimento, mais notavelmente o vídeo, que está previsto representar mais do que 50 por cento do tráfego realizado este ano [1].

Para além do grande crescimento do consumo da largura de banda, a NOS Madeira tem de fazer uma gestão dos recursos que possuí. A ligação para fora da Madeira por cabo submarino é alugada a terceiros e torna-se de facto indispensável ter um sistema que controla a largura de banda de forma eficiente.

A NOS Madeira tem, desde 2000, um sistema interno próprio em que, a cada utilizador, é atribuído um nível de prioridade de acordo com tráfego efetuado. Os níveis de prioridade são de 1 a 7 e os clientes que efetuem mais tráfego acabam por ter menor prioridade ao longo do tempo, resultando em menos velocidade, do que os clientes que efetuam menos tráfego. Se um utilizador deixar de realizar tanto tráfego então o nível irá aumentar de novo, ao longo do tempo.

1.1 Motivação e objetivos

O sistema que se encontrava em vigor corria sobre uma versão descontinuada do Windows, usando um misto de VBScript, *Data Transformation Packages* com importação dos níveis de prioridade sobre *File Transfer Protocol* (FTP). Isto significava pouca fiabilidade e não havia qualquer flexibilidade a nível das regras de negócio. A empresa tinha uma grande necessidade de substituir o sistema atual devido a problemas de escalabilidade, manutenção, modularidade e falta de integração com os sistemas de suporte de informação.

O novo sistema seria constituído por 3 subsistemas, uma colaboração entre 3 alunos da Universidade da Madeira, em que cada um implementaria um subsistema. Em primeiro lugar, um subsistema que recolhe e mapeia os dados de utilização, criado por Daniel Aguiar [2]. Em segundo, um subsistema de contabilidade que analisa esses dados, criado por Luís Ferreira [3]. Por fim, o subsistema controlador da rede, que aplica o resultado da análise dos dados. Este último subsistema, denominado daqui em diante por Servidor de Políticas (SP), é o objetivo deste trabalho. A principal função deste subsistema é fazer com que a análise feita pelo subsistema de contabilidade seja aplicada através de políticas, que são níveis de qualidade de serviço, comunicando com a máquina de rede que faz cumprir essas políticas, o *Service Control Engine* (SCE).

1.1.1 Proposta inicial

A proposta inicial passava por utilizar a *Cisco SCMS SCE Subscriber API*. Esta *Application Programming Interface* (API) permite realizar todas as operações de sincronização e gestão dos *subscribers*, comunicando diretamente com a máquina de rede. Para que isso fosse possível, teria de ser desenvolvido um programa, utilizando a linguagem Java, que utilizasse essa API. A comunicação com a máquina da rede seria realizada através de *Proprietary Remote Procedure Call* (PRPC), um protocolo proprietário da Cisco. Estava implicado, assim, construir todos os serviços web para que fossem compatíveis com os serviços usados nos outros subsistemas e os serviços de suporte já atualmente utilizados.

1.1.2 Proposta e objetivos adotados

Contudo, ainda numa fase inicial, foi proposta uma alternativa, pela empresa, que permitiria que a manutenção e futuras modificações ao Servidor de Políticas fossem bastante mais fáceis. Esta alternativa passa por usar um dos módulos de software do *Subscriber Manager* (SM), que comunica com o SCE. Este módulo, denominado por *Simple Object*

Access Protocol (SOAP) *Login Event Generator* (LEG), permite enviar dados sobre determinado IP (logo um cliente), como o seu nível de qualidade de serviço, através de uma interface SOAP, ou seja, um serviço web. Esta solução poderia ser então implementada de forma semelhante aos outros dois projetos, com recurso à linguagem PHP e a Zend Framework. Assim, como objetivos do trabalho, foram definidas as seguintes tarefas:

- Levantamento e compilação de requisitos para o subsistema;
- Desenho de modelos e diagramas do sistema a desenvolver;
- Implementação de acordo com os standards atuais;
- Colaboração com os autores dos dois outros subsistemas, com o propósito de integração entre estes;
- Configuração do *Subscriber Manager*;
- Realização de testes;
- Acompanhamento do sistema durante as fases de desenvolvimento (*staging*) e entrada em produção.

1.2 Estrutura do documento

O conteúdo deste trabalho está organizado por 5 capítulos, da seguinte forma:

- No capítulo 2, é primeiro descrito o sistema como um todo, e depois, mais pormenorizadamente o subsistema e sistemas externos nos quais o trabalho se foca. São descritos também os conceitos utilizados, nomeadamente os serviços web, usados na comunicação entre sistemas, SOAP, WSDL e a arquitetura MVC.
- No capítulo 3 são apresentados os requisitos, a modelação do sistema e diagrama da arquitetura.
- O capítulo 4 descreve a implementação, com menção da metodologia adotada, ferramentas, estrutura do projeto e aspetos gerais e a análise ao código que satisfaz os requisitos propostos.
- No capítulo 5 são mencionados os testes realizados assim como os resultados do trabalho.
- No capítulo 6 são apresentadas as conclusões finais do trabalho, em conjunto com sugestões de melhorias ou funcionalidades que poderiam ser realizadas no futuro.

- Nos anexos estão incluídos: o diagrama de Gantt com as diferentes fases do projeto; o guia de configuração do SM; o ficheiro WSDL da Cisco usado pelo sistema para comunicar com o SM.

Capítulo 2

Descrição do sistema

Neste capítulo é descrito o sistema como um todo, a sua composição e interações, internas e externas. São descritos também os grandes intervenientes deste trabalho, nomeadamente o servidor de políticas, o *Subscriber Manager* e o *Service Control Engine*. Por fim, são abordadas as diferentes tecnologias utilizadas.

2.1 Visão geral

A arquitetura geral do sistema baseia-se em 3 subsistemas que desempenham as funções de: recolha e mapeamento dos dados estatísticos (1), tratamento dos dados (2) e de cumprimento das políticas (3). A figura 2.1 apresenta todos os componentes. Devido à complexidade do sistema e para que fosse implementado com sucesso durante os prazos estipulados o trabalho foi repartido por 3 pessoas, numa colaboração, tal como foi mencionado no capítulo introdutório.

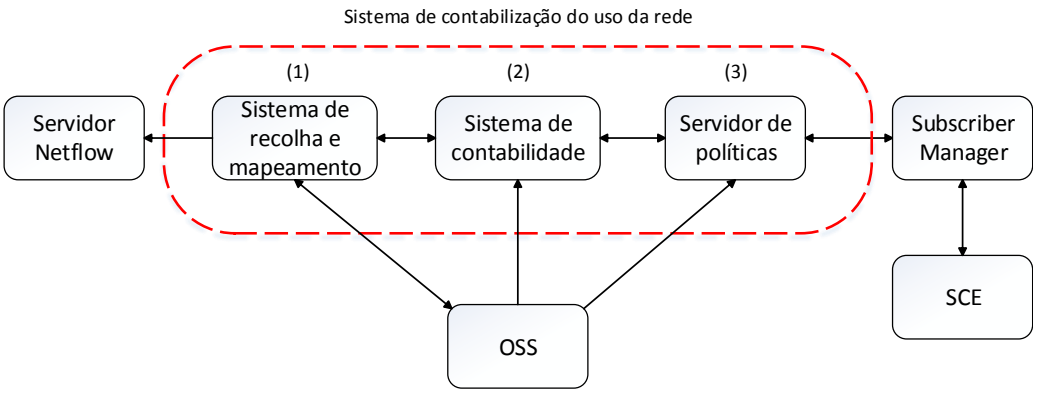

Figura 2.1: Diagrama geral do sistema de contabilização

Em primeiro lugar, o sistema de recolha e mapeamento é responsável pela recolha de dados do tráfego realizado por cada endereço IP. Esta recolha é realizada com recurso às máquinas de rede, nomeadamente o servidor de *Netflow*, onde é guardado, periodicamente, o total de tráfego feito em *bytes* de cada IP ativo. Esta recolha tem uma periodicidade de 15 minutos. Os dados são depois associados a clientes para poderem ser processados pelas regras de negócio, da responsabilidade do módulo denominado por Sistema de Contabilidade. Esta informação dos clientes é, em cada momento da recolha, retirada de um serviço interno, denominado por *Operations Support System* (OSS), ou seja, o sistema de suporte às operações. Este é o sistema que contém muitos dos serviços de suporte operacional à rede já existentes, como informação sobre que produtos é que os clientes possuem, estado e parâmetros dos modems entre outras informações.

Os clientes vêem a sua largura de banda aumentar ou diminuir de acordo com um sistema de prioridades que privilegia com maior largura de banda os utilizadores que efetuaram menos tráfego. Este novo sistema pretende trazer, também, alguma flexibilidade a nível da definição das regras de negócio. Há, assim, uma série de parâmetros que podem ser utilizados para a criação dessas regras [3]:

- Tráfego realizado – incluído no registo que está a ser calculado no momento;
- Tráfego efetuado anteriormente – com recurso aos registos previamente calculados;
- Período de tempo – data e hora da realização do tráfego;

O produto do cliente indica quais as regras de negócio que serão usadas, ou seja, como serão feitos os cálculos e quais os valores de QoS atribuíveis. Por exemplo: um determinado produto pode ter em conta o tráfego efetuado nas últimas duas semanas, que deve depois ser multiplicado por 20% e adicionado ao tráfego efetuado nas últimas 24 horas. Este valor final será utilizado para determinar o nível de prioridade que deverá ser atribuído ao cliente. As regras são definidas pela empresa de acordo com as suas necessidades.

O produto final do processamento das regras deve ser um nível de prioridade, denominado também por *package*, para cada um dos endereços MAC (*Media Access Control*) processados. Embora o informação enviada para o Servidor de Políticas inclua apenas os endereços IP, o *package* fica antes associado a um endereço MAC. Esse *package* é que define os valores precisos sobre quanta largura de banda é dada a cada endereço IP. Há também a possibilidade de incluir certas exceções durante o tratamento dos dados para que determinados endereços IP ou MAC tenham sempre uma determinada largura de banda garantida. Aqui incluem-se os casos de contratos com instituições ou empresas que requerem dessa necessidade.

Por fim, e onde se centra este trabalho, esses *packages* têm de ser aplicados. Para que isso seja possível, é necessário comunicar com outro servidor, o *Subscriber Manager* que depois irá sincronizar essa informação automaticamente com outra máquina de rede, o *Service Control Engine* (SCE), que aplica as políticas de acordo com os *packages* definidos.

2.2 Servidor de políticas (SP)

O servidor de políticas terá a capacidade de processar pedidos vindos do *Subscriber Manager* (SM) para IPs que este não conhece (*pull*) e de processar atualizações vindas do subsistema de contabilidade (*push*). É de notar que, quando se fala em IPs, no âmbito deste trabalho, também se incluí IPs de sub-redes, ou seja, existe a capacidade de aplicar níveis de prioridade a uma sub-rede inteira ou a um único endereço IP. Para melhor exemplificar os casos acima descritos, seguem-se alguns diagramas.

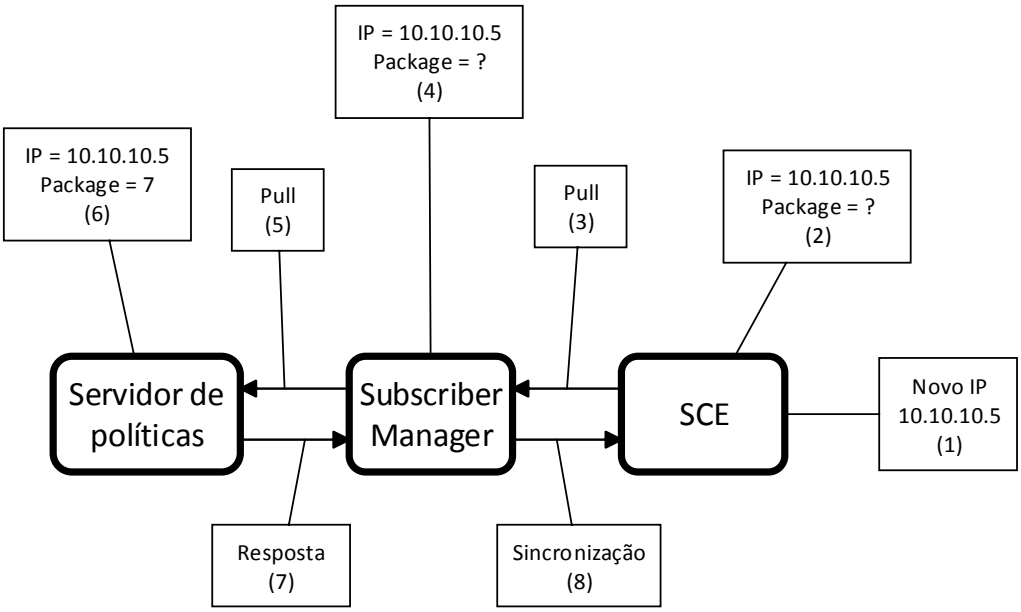

Figura 2.2: Caso em que o SCE não conhece o novo IP

No exemplo da figura 2.2 há tráfego a ser feito a partir de um novo IP (1), cujo package o SCE não conhece (2). É enviado então um pedido (3) ao SM a perguntar por esse IP. Se o SM também não tiver nenhum registo desse IP (4), então o servidor de políticas (SP) será questionado (5). Neste caso, existe um registo para esse IP na base de dados do SP e este irá responder adequadamente ao pedido (7). Por fim, assim que o SM receber essa informação, irá realizar a sincronização automaticamente com o SCE (8).

Poderá acontecer que o SP não conheça também esse IP, caso ilustrado na figura 2.3.

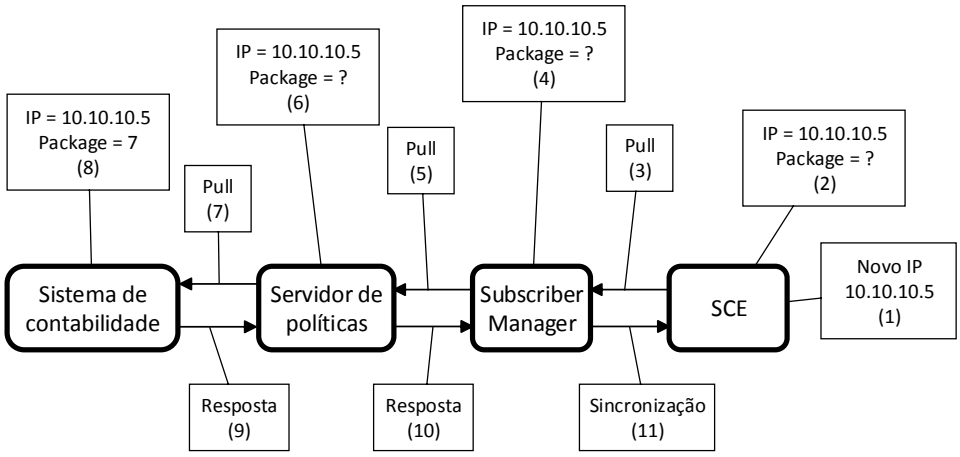

Figura 2.3: Caso em que apenas o sistema de contabilidade conhece o *package*

Neste caso existe mais um pedido, que é realizado ao sistema de contabilidade (7), na hipótese de este já ter informação sobre esse IP (8). Se isso não se verificar, então a resposta dada ao SM (9, 10) não irá incluir nenhum *package*, pelo que o IP ficará inserido no *package* por omissão.

Por fim, para o *push*, quando existe nova informação vinda do sistema de contabilidade, essa deve ser processada e transmitida ao SM, como mostra a figura 2.4.

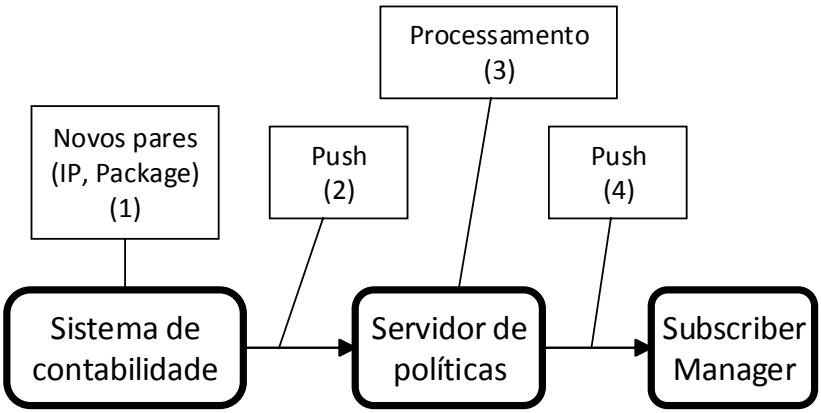

Figura 2.4: Caso em que existe nova informação do sistema de contabilidade

Os casos acima demonstrados representam a forma como são realizados os pedidos *push* e *pull* do sistema proposto. Existem, contudo, outras funções realizadas pelo servidor de políticas, como a disponibilização de serviços web e adição de *overrides*. *Override* é apenas uma sobreposição de um nível de prioridade para um IP, ou sub-rede, durante um determinado período de tempo. Mais detalhes sobre estas funcionalidades serão apresentados no capítulo 3.

2.3 Service Control Engine (SCE)

A parte principal do trabalho passa por interagir com o *Service Control Engine* (SCE), que é uma máquina de rede. Esta interação é, de certa forma, realizada indiretamente, com recurso ao *Subscriber Manager* (SM).

O SCE tem a capacidade de realizar *Stateful Deep Packet Inspection* (SDPI). Este tipo de inspeção atua na camada de aplicação do modelo OSI, em que todo o fluxo individual de tráfico e o estado de cada sessão é reconstruido pelo SCE. A gestão desta informação permite que a plataforma possa identificar aplicações que ativam portas de forma dinâmica ou que envolvam vários fluxos de dados que estão inter-ligados, normalmente encontrados em comunicações do tipo *voice-over-ip* (VoIP), voz através da Internet, e outros protocolos relacionados com reprodução ao vivo de multimédia. Esta capacidade de inspeção permite um controlo mais refinado, e em tempo real, do tráfego que passa pela rede [4].

Este controlo da largura de banda pode ser aplicado a um determinado IP, em centenas de diferentes protocolos da camada de aplicação como o HTTP, FTP, RTMP, SMTP, entre outros. Isto significa, por exemplo, que para determinada política, um utilizador pode utilizar 5 *megabytes* por segundo (MB/s) de banda para tráfego HTTP mas ao mesmo tempo só poder utilizar 1MB/s para tráfego RMTP. O SCE é assim capaz de analisar e classificar o tráfego que entra ou sai e limitá-lo de acordo com as políticas em vigor para esse utilizador.

Esta capacidade de inspecionar o tráfego permite também que, por exemplo, independentemente do tráfego realizado por um cliente, exista um mínimo de banda larga garantido para que as chamadas de VoIP tenham uma qualidade admissível, entre outras aplicações práticas, se tal for relevante para o negócio da empresa.

De modo a poder fazer esta gestão, o SCE fica, então, entre o dispositivo agregador, como um CMTS (*Cable Modem Termination System*), e a Internet. Este dispositivo agregador trata de converter o sinal dos modems em pacotes IP. Para melhor exemplificar, a figura 2.5 apresenta um esquema genérico, simplificado e adaptado de [4]:

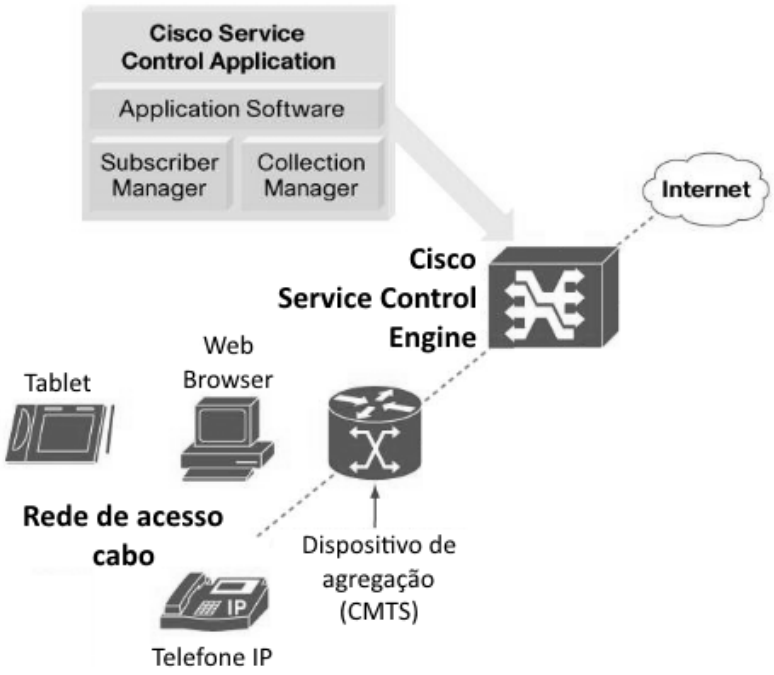

Figura 2.5: Exemplo de um cenário de utilização do SCE

Uma outra característica importante que o SCE tem é a possibilidade de se aplicar um *package* a toda uma sub-rede. Ou seja, o SCE irá dividir equitativamente a largura de banda disponível por todos os IP's dessa sub-rede que se encontram ativos.

O presente trabalho foi desenvolvido tendo em conta que, em produção, seria usada a versão 8000 do SCE.

Agora que foram apresentadas mais em pormenor as funções realizadas pelo SCE, podemos passar ao componente que realiza as operações de sincronização dos dados relativos aos *subscribers* (clientes), o *Subcriber Manager*.

2.4 Subscriber Manager (SM)

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 [5].

A informação de um *subscriber* entra no SM numa de duas maneiras:

- Automaticamente quando o *subscriber* está em linha – Um *Login Event Generator* (LEG) identifica um evento de login do *subscriber* e envia-o para o SM;

- Manualmente – Um fornecedor de serviço importa a informação dos *subscribers* para o SM a partir de um ficheiro, ou através do uso das *Command-Line Utilities* (CLU).

O SM pode passar informação automaticamente para a plataforma SCE. Alternativamente, a informação pode ficar na base de dados do SM até a plataforma SCE a requerer.

A nível deste trabalho, o SM realiza o seguinte conjunto de funções:

- Guarda toda a informação dos *subscribers* numa base de dados interna;
- Serve como intermediário entre o servidor de políticas e o hardware (SCE). Realiza a sincronização da informação relativa aos *subscribers* entre o SM e o SCE;
- Fornece a interface de comunicação em SOAP para a introdução de novos dados sobre os *subscribers*.
- Pergunta ao servidor de políticas sobre o *package* de um endereço IP quando este não se encontra na sua base de dados.

O SM é essencial para a solução adotada, visto que é o responsável por fornecer o serviço web (SOAP LEG) que permite realizar as atualizações dos dados relativos aos *subscribers* – o endereço IP e o *package* correspondente.

2.5 Conceitos importantes

Esta secção aborda alguns conceitos importantes que convém mencionar para melhor compreender o restante trabalho.

2.5.1 Serviços web

Os serviços web são sistemas de software desenhados para suportar comunicações interoperáveis entre máquinas sobre uma rede [6]. Isto significa que pode haver um agente criado para consumir determinado serviço numa determinada linguagem e para outro serviço um agente criado numa linguagem completamente diferente. Para que esta comunicação aconteça são definidas interfaces que especificam o modo de como deve ser efetuada a comunicação. Neste trabalho essas interfaces são especificadas em documentos que seguem a *Web Services Description Language* (WSDL) e a comunicação é realizada através de SOAP. As secções que se seguem abordam diretamente o SOAP e a WSDL.

Simple Object Access Protocol (SOAP)

O SOAP fornece um mecanismo de troca de informação estruturada entre *peers* numa rede de computadores, de forma descentralizada, usando XML. Esta estruturação e definição dos tipos de dados a serem transmitidos não é algo que seja forçado pelo protocolo; a semântica depende simplesmente dos detalhes da implementação. O que o SOAP oferece é uma espécie de base, um conjunto de regras para a especificação de tipos de dados e convenções para representar pedidos e respostas [7].

O protocolo só por si não define o modo de transmissão e depende assim de outros protocolos de transporte, usualmente o HTTP, usado nos *web services*. Segue-se um exemplo de uma mensagem SOAP dentro de um pedido HTTP:

```
1  POST /OSS HTTP/1.1
2  Host: policyserver.localdomain
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5  SOAPAction: "Algum-URI"
6
7  <SOAP-ENV:Envelope
8    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
9    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
10    <SOAP-ENV:Body>
11        <m:GetPairs xmlns:m="Some-URI"></m:GetPairs>
12    </SOAP-ENV:Body>
13 </SOAP-ENV:Envelope>
```

Esta independência dos protocolos acaba por ser uma vantagem, pois há a flexibilidade de escolher o protocolo de comunicação que for mais conveniente para quem desenvolve o sistema.

Web Services Description Language (WSDL)

WSDL é uma forma de descrever serviços web em formato XML. Esses serviços são descritos como um conjunto de pontos de acesso para troca de mensagens. Serve, essencialmente, como uma receita para automatizar os detalhes que envolvem a comunicação entre sistemas ou aplicações [8]. Detalhes como - qual o nome do serviço, quais os os métodos oferecidos, o tipo de parâmetros e os valores a receber.

Um documento WSDL é assim um conjunto de definições. Segue-se um exemplo retirado deste trabalho, usado para comunicação com o sistema de contabilidade:

```
1  <definitions ...>
2      <types>...</types>
```

```
3     <portType name="Accounting_Model_ServicesPort">...</portType>
4     <binding name="Accounting_Model_ServicesBinding" type="
          tns:Accounting_Model_ServicesPort">...</binding>
5     <service name="Accounting_Model_ServicesService">...</service>
6     <message name="pushBulkIn">
7         <part name="pairs" type="soap-enc:Array"/>
8     </message>
9     <message name="pushBulkOut">
10        <part name="return" type="xsd:boolean"/>
11    </message>
12 </definitions>
```

Neste exemplo é relativamente fácil inferir que são transmitidos dois tipos de mensagens relativas ao método `pushBulk`: uma que recebe um *array* de pares (`pushBulkIn`), que é enviada pelo sistema de contabilidade, e outra que recebe um *boolean* (`push BulkOut`), que é a resposta do servidor de políticas.

2.5.2 Model–view–controller

O padrão arquitetural de software *Model-view-controller* (MVC) divide um programa em três componentes lógicos que interagem entre si [9]:

- **Model** - O modelo trata de gerir o comportamento e os dados pertencentes ao domínio da programa. Tem de responder a pedidos sobre o estado desses dados, normalmente provenientes da vista e responde também a pedidos de mudança de estado, normalmente provenientes do controlador.

- **View** - A vista define e gere como é que os dados devem ser apresentados ao utilizador.

- **Controller** - O controlador é quem gere as interações do utilizador com o programa. Estas interações podem significar o envio de comandos ao modelo, para que este atualize o seu estado, ou então, comandos para a vista, para que esta mude a maneira de apresentar os dados.

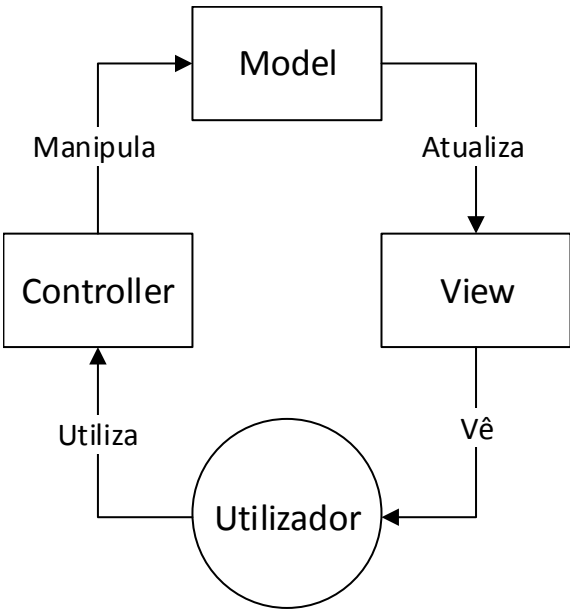

Figura 2.6: O padrão arquitetural MVC

Este padrão é bastante comum em aplicações orientadas à Internet. É normalmente utilizado quando existem diversas maneiras de se ver e interagir com os dados, e quando os requisitos futuros de interacção e apresentação desses dados são desconhecidos [10]. Razão esta pela qual se compreende que este padrão seja usado numa ferramenta de software como a Zend Framework, que tenta ser o mais versátil possível.

2.6 Conclusão

Neste capítulo foi descrito cada um dos componentes do sistema, entrado mais em pormenor sobre o Servidor de Políticas, a plataforma SCE, o *Subscriber Manager* e alguns conceitos importantes. O próximo capítulo aborda a especificação e modelação do sistema.

Capítulo 3

Especificação e modelação do sistema

Antes de se passar à implementação existe uma razão fulcral para a criação de modelos: para que possamos entender melhor o sistema que estamos a desenvolver. Segundo [11] existem quatro objetivos que se atingem com a modelação: ajuda à visualização de como o sistema é, ou como queremos que seja; permite especificar a estrutura ou o comportamento de um sistema; serve como um guia durante a construção do sistema; e, finalmente, como uma forma de documentar as decisões tomadas.

Neste capítulo são inicialmente descritos os requisitos do trabalho. Depois, são apresentados alguns modelos que proporcionam uma perspetiva abstrata sobre partes importantes do novo sistema. São modelos usuais, elaborados numa notação gráfica baseada na *Unified Modeling Language* (UML).

Posteriormente, são apresentados os diagramas de casos de utilização onde são identificadas as interações externas com os utilizadores do sistema. Incluída está também uma tabela que complementa com mais detalhes cada caso. De seguida, com recurso aos diagramas de atividades, é especificado o fluxo dos processos mais significativos. Para os dados armazenados no sistema é apresentado um modelo de dados. Por fim, é apresentado o diagrama da arquitetura, com os componentes e fluxo de dados.

3.1 Requisitos

A definição de requisitos foi um dos primeiros passos do trabalho. Tratou-se de um processo iterativo entre o aluno e a empresa para compreender não só as necessidades do subsistema Servidor de Políticas, mas de todo o sistema de contabilização. É sempre importante investir o tempo suficiente para entender as necessidades do projeto e após várias reuniões os requisitos foram fechados da forma que se apresenta de seguida.

3.1.1 Requisitos funcionais

Os requisitos funcionais são declarações de serviços ou tarefas que o sistema deve disponibilizar ou realizar e como deve reagir à introdução de nova informação ou a outras situações mais em específico [10]. Os requisitos funcionais deste trabalho foram os seguintes:

1. O sistema deve ser capaz de fazer cumprir níveis de qualidade de serviço no *Cisco Service Control Manager* (SCE). Para tal, são usadas regras que se baseiam em políticas de negócio, definidas como *packages*, e a aplicação destas a determinados IP's.
   (a) Este sistema depende de um componente já existente, o *Subscriber Manager* (SM), que tem de ser configurado de modo a servir como intermediário (*proxy*) entre o servidor de políticas e o SCE.
   (b) O sistema deve ser capaz de comunicar com o SM de modo a:
      i. Enviar resposta aos pedidos de especificação de um *package* para determinado IP.
      ii. Enviar conjuntos de pares [IP, Package].
   (c) O sistema deve ser capaz de comunicar com o sistema de contabilidade de forma a:
      i. Obter o nível de qualidade de serviço (*package*) para um determinado IP (*pull*).
      ii. Poder receber nova informação não solicitada, na forma de pares [IP, Package] (*push*).
   (d) O sistema deve ser capaz de substituir as políticas aplicadas a um IP de forma volátil.
      i. Deve ser possível a especificação de um período de tempo de atuação para esses *overrides*.
2. O sistema deve ser capaz de disponibilizar serviços web.
   (a) Através do protocolo SOAP.
   (b) A descrição dos serviços oferecidos deve ser feita e disponibilizada utilizando a WSDL (*Web Services Description Language*).
   (c) Os serviços a disponibilizar são:
      i. Consulta dos *packages* dado um ou mais IP's.

ii. Consulta dos IP's associados a um *package*.
iii. Consulta da quantidade de IP's associados a um *package*.
iv. Consulta das quantidades de IP's associados a todos os *packages* existentes.
v. Consulta dos *overrides* temporários, (a) dado um IP ou (b) na totalidade.
vi. Adicionar *override* temporário a um determinado IP.
vii. Remover *override* temporário de um determinado IP.

3. O sistema deverá comunicar com uma base de dados relacional MySQL para armazenamento de dados e *caching*.
   (a) *Caching* dos pares [IP, Package].
   (b) Armazenamento dos *overrides* temporários.
4. Envio de notificações (*traps*) SNMP como mecanismo de alarme para eventos ou falhas importantes.

3.1.2 Requisitos não-funcionais

Os requisitos não-funcionais são restrições impostas nos serviços ou nas tarefas oferecidas pelo sistema. Podem incluir restrições a nível de tempo, na forma como o projeto deve ser desenvolvido, ou simplesmente a inclusão de standards. Geralmente aplicam-se ao sistema como um todo [10]. Seguem-se então os requisitos não-funcionais do trabalho:

1. O sistema será desenvolvido para a plataforma UNIX.
2. O sistema será compatível e implementado com a versão 1.12.x da *Zend Framework*.
3. O sistema deverá ter, durante o normal funcionamento, um tempo de resposta máximo de 3 segundos a qualquer pedido dos serviços web disponibilizados.
4. Qualquer falha de comunicação com o sistema de contabilidade ou com o SM não deve impedir o sistema de continuar outras operações que necessite de realizar.
5. O sistema deverá ser implementado de forma a estar constantemente funcional, sendo esperada normalmente uma disponibilidade de 99.98%.
6. Após uma falha crítica do sistema é expectável que este seja reiniciado e esteja funcional até 2 minutos após esta.
7. O código fonte deve estar comentado de forma a que seja fácil extrair o que é suposto cada método ou módulo fazer.

3.2 Casos de utilização

De modo a melhor representar todas as interações do sistema com os atores e sistemas externos segue-se, na figura 3.1, o diagrama de casos de utilização onde entram como intervenientes o subsistema de contabilidade, o SM e o utilizador OSS.

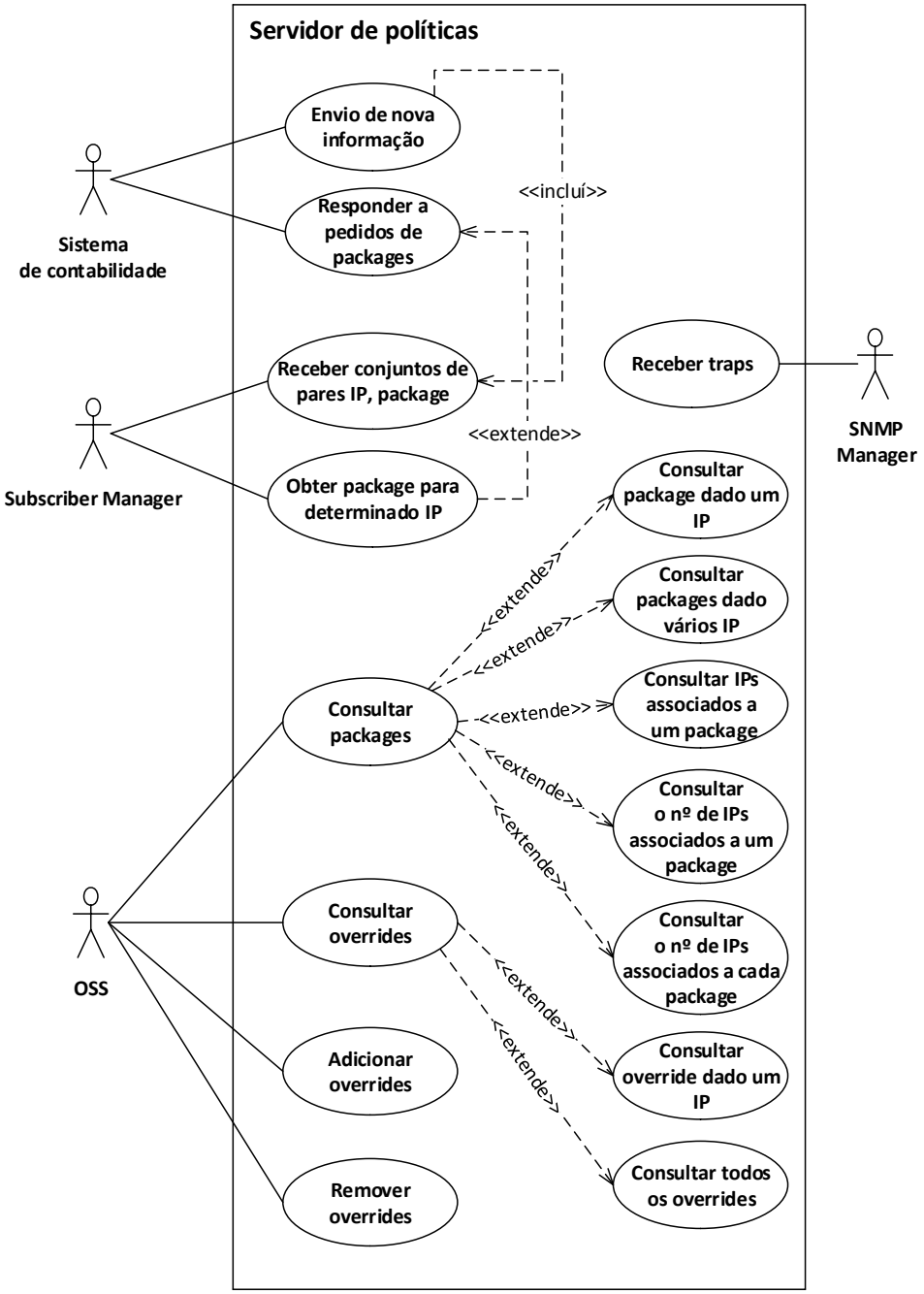

Figura 3.1: Diagrama de casos de utilização

O diagrama só por si não contém quaisquer detalhes ou contexto sobre cada um dos casos. Sendo assim e como é proposto em [10], é apresentada na tabela 3.1 com os atores, descrição, dados, estímulos e respostas para cada um dos casos da figura 3.1. Qualquer omissão de um dos campos de uma especificação é porque, ou faz parte de uma generalização (e é omitida para se evitar repetição), ou não é aplicável.

Tabela 3.1: Especificação dos casos de utilização

Caso	Envio de nova informação
Actor	Sistema de contabilidade
Descrição	Sempre que o sistema de contabilidade processa dados esses são enviados para o servidor de políticas. Esta informação, numa fase posterior, é usada para atualizar os registos no SM.
Dados	Pares IP, *package*.
Estímulo	*Push* do sistema de contabilidade.
Resposta	Confirmação do *push*.

Caso	Responder a pedidos de *packages*
Actor	Sistema de contabilidade
Descrição	Quando o sistema não tem informação do *package* que deve atribuir a determinado IP, este pede essa informação ao sistema de contabilidade, que, por sua vez, responde a esse pedido.
Dados	*Package*
Estímulo	Resposta ao pedido de *package*
Resposta	Confirmação de resposta.

Caso	Consultar packages
Actor	Utilizador OSS
Descrição	O utilizador que acede aos serviços do sistema tem a capacidade de consultar informação relativa aos IP's e aos *packages*. Para isso tem de especificar qual o tipo de consulta pretende e os respectivos parâmetros. O sistema trata então de realizar quaisquer cálculos necessários e de retornar a informação pedida.
Estímulo	Pedido do utilizador OSS.

Caso	Consultar package dado um IP
Descrição	Se o utilizador enviar um endereço IP neste tipo de consulta, como resposta recebe o *package* que lhe está associado ou a falta de qualquer associação.
Dados	Endereço IP
Resposta	*Package* ou nada.

Caso	Consultar packages dados vários IP
Descrição	Se o utilizador enviar vários endereços IP neste tipo de consulta, como resposta recebe os *packages* associados aos IP's fornecidos, ou a falta destes.
Dados	Endereço IP
Resposta	Conjunto de [IP, *package* ou nada].

Caso	Consultar IP's associados a um package
Descrição	Se o utilizador enviar um *package* neste tipo de consulta então serão retornados todos os IP's associados a este.
Dados	*Package*
Resposta	Conjunto de endereços IP.

Caso	Consultar o nº de IP's associados a um package
Descrição	Ao receber este pedido o sistema recolhe todos os registos associados ao *package* fornecido pelo utilizador e retorna esse valor.
Dados	*Package*
Resposta	Número de IP's associados.

Caso	Consultar o nº de IP's associados a cada package
Descrição	Ao receber este pedido o sistema recolhe todos os registos presentes e realiza a contagem do número de IP's associados a cada *package* e retorna estes dados ao utilizador.
Resposta	Conjuntos de [*package*, contagem]

Caso	Adicionar *overrides*
Actor	Utilizador OSS
Descrição	O utilizador pode realizar um pedido para ser adicionado um *override* temporário de um *package*a um determinado IP até certa data limite. Se houver um registo em vigor o sistema retornará um erro.
Dados	*Package*, endereço IP e data limite.
Estímulo	Pedido do utilizador OSS.
Resposta	Confirmação de pedido ou erro.

Caso	Remover *overrides*
Actor	Utilizador OSS
Descrição	O utilizador pode realizar um pedido para remover um *override* de um determinado IP.
Dados	Endereço IP
Estímulo	Pedido do utilizador OSS.
Resposta	Confirmação de pedido.

Caso	Consultar *overrides*
Actor	Utilizador OSS
Descrição	O utilizador que acede aos serviços do sistema tem a capacidade de consultar informação relativa aos *overrides*. Da mesma forma que é realizada na consulta dos *packages* tem de se especificar qual o tipo de consulta pretendida e os respectivos parâmetros.
Estímulo	Pedido do utilizador OSS.

Caso	Consultar *overrides* dado um IP
Descrição	Ao realizar este pedido o utilizador tem de fornecer um endereço IP. O sistema procura por um registo de *override* temporário, que esteja em vigor, com esse IP e retorna os dados deste.
Dados	Endereço IP

Resposta	Dados do *override*: IP, *package* e data limite.

Caso	Consultar todos os *overrides*
Descrição	Ao receber este pedido o sistema agrupa todos os registos de *overrides* temporários e envia-os ao utilizador.
Resposta	Todos os *overrides* temporários (vigor/expirados?)

Caso	Receber conjuntos de pares IP, package
Actor	SM
Descrição	De modo a poder manter-se atualizado, o SM permite receber nova informação relativa ao nível de QoS dos subscritores. Esta informação é enviada pelo sistema logo que possível.
Dados	Conjuntos de pares IP, *package*.
Estímulo	Nova informação relativa aos subscritores disponível.
Resposta	Confirmação de entrega.

Caso	Obter package para determinado IP
Actor	SM
Descrição	Sempre que o SM não tem informação sobre que *package* aplicar a determinado endereço IP, é feito um pedido ao sistema para que seja definido.
Dados	*Package*
Estímulo	Pedido do SM.
Resposta	Envio do *package*.

Caso	Receber traps
Actor	SNMP Manager
Descrição	No caso de falhas graves o sistema poderá enviar *traps* como forma de alarme, pelo que serão um ou mais *SNMP Managers* a recebê-las.
Dados	*SNMP Trap* específica do problema
Estímulo	Falha grave do sistema.

Resposta Envio da *trap*.

3.3 Diagramas de atividades

Como complemento aos próprios casos de utilização, é importante entender os principais processos do sistema e o fluxo de controlo de uma atividade para outra. Para além dos processos, são também considerados os dados e os sistemas envolvidos.

O primeiro caso (figura 3.2) descreve o que acontece quando o *Subscriber Manager* recebe um pedido *pull* de um IP que não conhece.

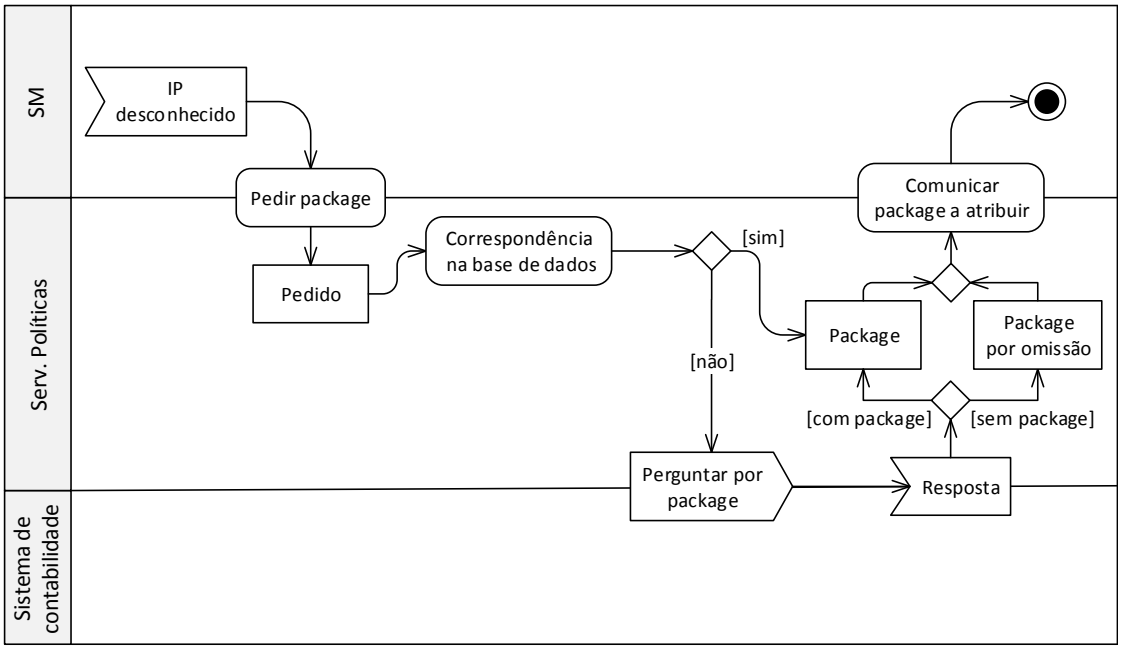

Figura 3.2: Diagrama de atividade para um IP desconhecido

Inicialmente, é lançado um pedido por parte do SM que chega ao servidor de políticas. Depois de esse IP ser inspecionado, poderá ou não haver uma correspondência com a informação existente na base de dados. Se não existir, o servidor de políticas pergunta ao servidor de contabilidade na esperança dessa informação já estar disponível. Se não for retornado um package, então o IP será colocado no package por omissão.

O próximo cenário, representado na figura 3.3, acontece quando o sistema de contabilidade tem novos dados para enviar.

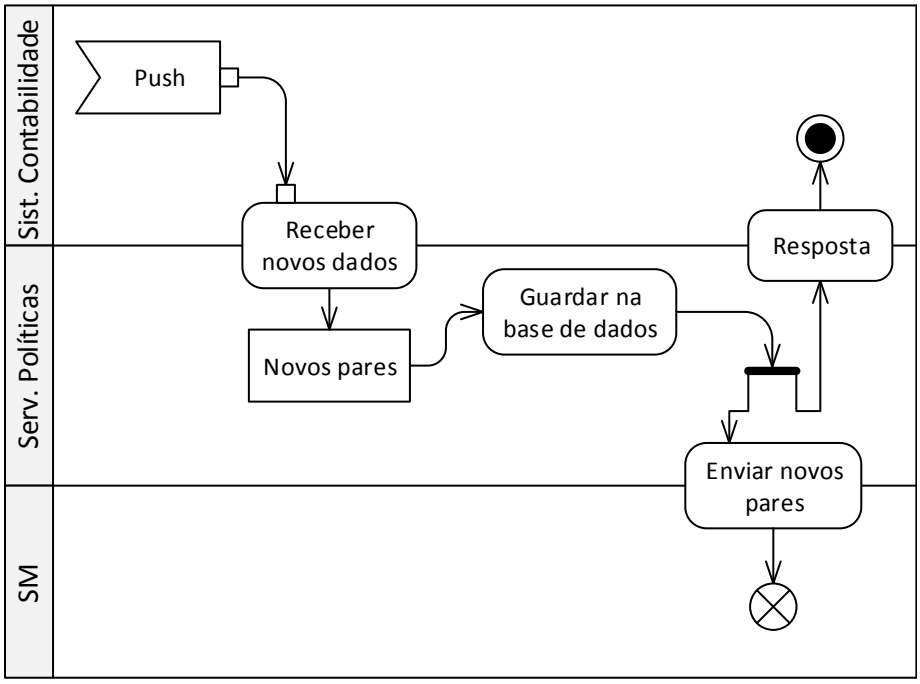

Figura 3.3: Diagrama de atividade no momento em que se recebem novos dados

O fluxo de controlo e dados nesta parte é relativamente simples. Tudo o que o servidor de políticas precisa de fazer é guardar os novos pares (IP, package) e lançar um processo que trata de os enviar, assincronamente. Existem, assim, dois fluxos, embora um acabe imediatamente assim que é lançado o processo externo. O principal objetivo desta separação é para que a resposta ao sistema de contabilidade não esteja dependente do envio dos pares. Esta é, claro, uma representação em alto nível, pois o processo que realiza o envio dos pares terá de ser mais detalhado.

A figura 3.4 apresenta o fluxo de um pedido SOAP proveniente do OSS. As operações presentes referem-se ao ponto 2 dos requisitos funcionais, apresentados na secção 3.1.1. Todos os serviços disponibilizados seguem a mesma lógica, havendo apenas diferenças nas *queries* realizadas à base de dados e na modificação de *overrides* temporários, onde está também envolvido o SM.

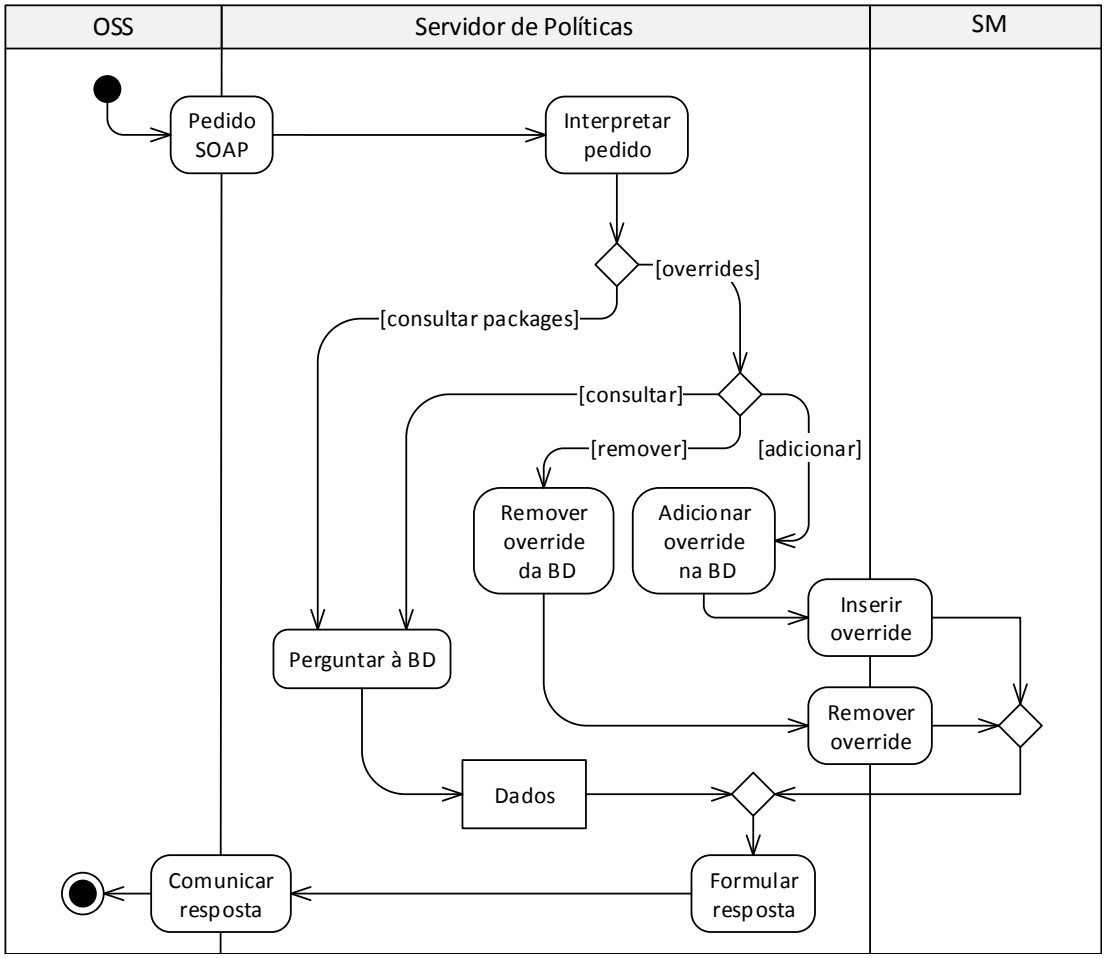

Figura 3.4: Diagrama de atividade para um pedido SOAP

Agora que já se deu uma ideia geral dos principais processos do sistema, vamos passar a descrever o modelo de dados.

3.4 Modelo de dados

O requisito funcional nº. 3 especificava que o armazenamento da informação deveria ficar num Sistema de Gestão de Base de Dados Relacional (SGBDR), nomeadamente num servidor MySQL. É comum e adequado, que seja realizado um simples modelo Entidade-Relação que especifique os tipos, tamanho dos dados e possíveis relações entre eles. Este modelo está presente na figura 3.5, que se segue.

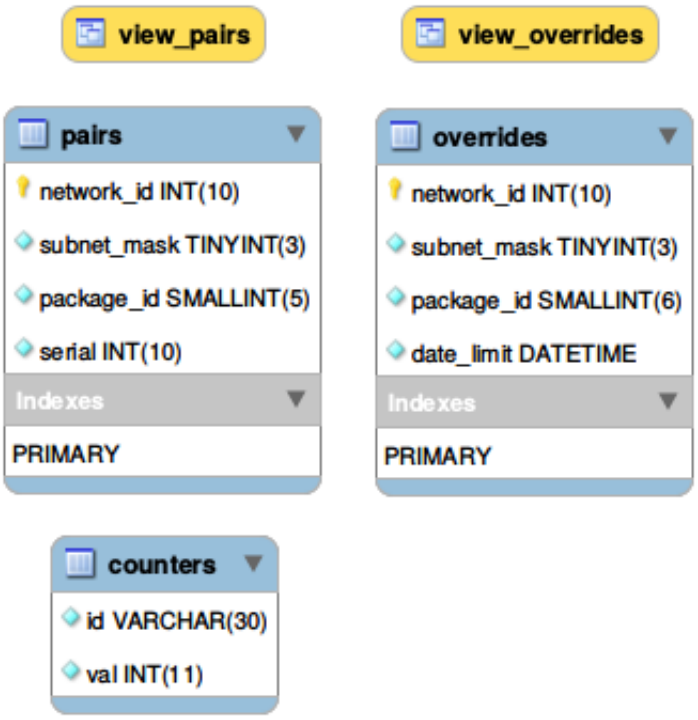

Figura 3.5: Modelo de Entidade-Relação (ER)

De seguida, são descritas as tabelas presentes neste modelo.

Tabela *pairs* O grande foco deste sistema são os pares de dados IP e package. A informação sobre esses pares é armazenada na tabela *pairs*. Para cada endereço IP a (network_id), que incluí também a máscara de sub-rede (subnet_mask), existe sempre um identificador do package a que este pertence (package_id).

Embora fosse possível manter IP e máscara juntos, esta separação entre a máscara e o IP permite que o processamento e a procura sejam mais eficientes. Não há, assim, necessidade de separar o IP da máscara cada vez que é necessário realizar uma consulta por qualquer um dos campos. Um outro ponto, ainda dentro da eficiência, foi a representação decimal (INT) do endereço IP em vez da notação decimal com pontos usando caracteres (VARCHAR), o que permite uma procura mais rápida. Se fossem usados caracteres teria de haver uma comparação *byte* a *byte* até ser encontrada uma diferença. Esta representação em decimal permite que, para além de ser necessária apenas uma simples operação do processador para se compararem dois IP's, seja possível também utilizar operações de deslocamento de bits (*shift*) que tornam o processo de verificar se um IP está dentro de uma sub-rede mais célere.

Existe também outra vantagem para a representação em decimal. O próprio MySQL contém funções de conversão [12] de e para decimal fazendo com que, no futuro, com

a adoção do IPv6, seja simples incluir essa nova versão do protocolo no sistema. Em termos práticos, para o utilizador dos serviços web, esta conversão é transparente, sendo que a representação aparece na forma esperada, usando a notação decimal em grupos de octetos separados com pontos. Um endereço IP nesta notação, como por exemplo, `172.16.0.1`, na base de dados ficará representado como `2886729729`. O cálculo para esta representação em decimal é realizado com base na seguinte fórmula:

$$w \times 256^3 + x \times 256^2 + y \times 256 + z \tag{3.1}$$

em que w, x, y, z representam cada um dos grupos de octetos, do mais significativo para o menos significativo.

Ainda incluído na tabela *pairs*, encontra-se um número de série (serial) que serve apenas como referência aquando a entrega de novos pares por parte do sistema de contabilidade. Mais informação relacionada com este número será abordada no capítulo da implementação.

Tabela *counters* Esta tabela serve simplesmente para salvaguarda de chaves e valores. A única utilização desta tabela deve-se ao valor do *serial* referido na tabela anterior, para manter registo e assim poder gerar novos valores para este.

Tabela *overrides* Por fim, a tabela dos *overrides* contém a mesma informação que se encontra na tabela *pairs*, com excepção da coluna *serial*, que, neste caso, não é relevante. Em vez disso, devido à natureza dos *overrides*, é a data limite que tem interesse, de modo a que se saiba até quando é que esse *override* deve ser aplicado.

Vistas Para facilitar a visualização dos IP's em situações em que é necessário consultar a base de dados de forma manual para, por exemplo, realizar operações que se encontram fora do âmbito dos serviços web disponibilizados, ou impossibilidade de utilizar os ditos serviços, são incluídas também duas vistas, `view_pairs` e `view_overrides`. Nestas vistas todos os IP's são apresentados na notação decimal com pontos. A apresentação das restantes colunas mantém-se idêntica.

3.5 Diagrama da arquitetura

O diagrama da arquitetura, apresentado na figura 3.6, mostra a divisão do sistema numa perspectiva macro, demonstrando também o fluxo de dados entre os diversos intervenientes. É de destacar a divisão entre os módulos dos serviços web e o módulo que realiza o *push* dos pares para o SM. Isto deve-se ao facto do módulo ser independente do servidor Web, tendo o potencial de poder ser invocado por qualquer processo.

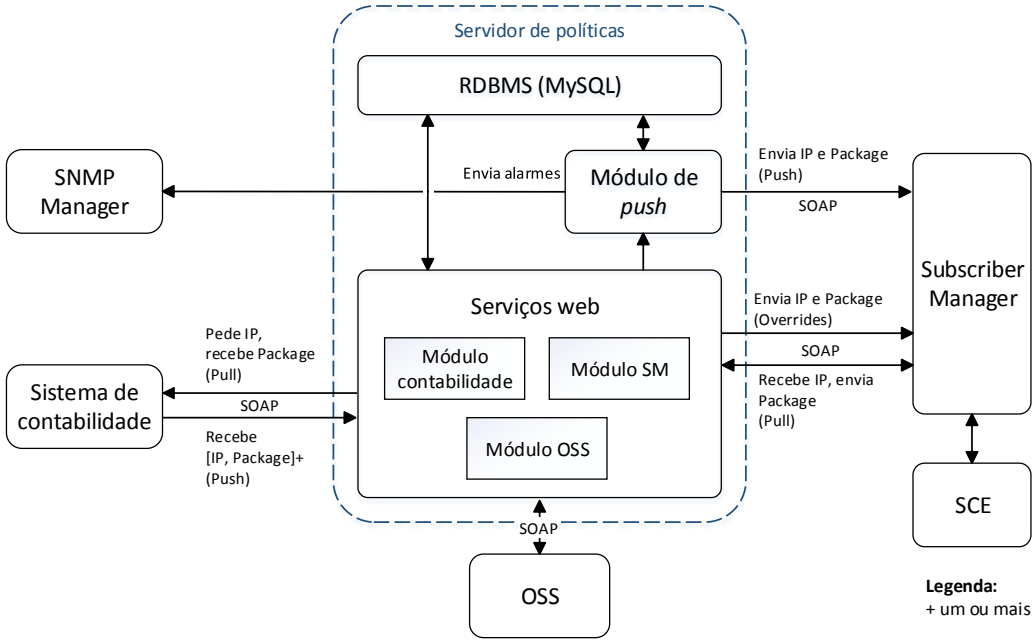

Figura 3.6: Diagrama da arquitetura com os diversos componentes e fluxo de dados

De seguida, apresentam-se as funções que cada componente desempenha em relação os requisitos propostos:

- o módulo contabilidade incluí a lógica por detrás do ponto 1-c dos requisitos funcionais
- o módulo OSS incluí a lógica por detrás dos pontos 1-d, 2-c dos requisitos funcionais
- o módulo SM incluí a lógica por detrás do ponto 1-bi dos requisitos funcionais

3.6 Conclusão

Este capítulo definiu as especificações e modelos que serão utilizados na implementação, tornando também mais óbvia a separação de responsabilidades de cada módulo e como é que estes devem ser desenvolvidos. O próximo capítulo irá abordar o processo, e detalhes, da implementação do sistema.

Capítulo 4

Implementação

Agora que os requisitos e os modelos estão definidos podemos, então, passar para a implementação. Neste capítulo é descrita a metodologia adotada, os ambientes de desenvolvimento, redundância, ferramentas utilizadas, seguida da estrutura geral do projeto fazendo referência também às suas configurações. Posteriormente é abordada a implementação de todos os módulos dos serviços web. Por fim, é explicada toda a lógica por detrás do módulo de *push*.

4.1 Metodologia

Este trabalho seguiu uma metodologia relativamente semelhante, mas não estrita, de um desenvolvimento orientado às funcionalidades. Houve também um esforço por fechar os requisitos principais assim que possível para que servisse como um contrato entre o autor e a empresa.

O desenvolvimento orientado às funcionalidades consiste em pequenas iterações, com ênfase na qualidade, incluindo resultados demonstráveis ao longo do processo [13]. Normalmente é desenvolvido um modelo geral; seguido da criação de uma lista das funcionalidades pretendidas; planeamento de cada funcionalidade, desenho e implementação de cada uma. Após ser terminada uma funcionalidade esta passa a fazer parte do produto principal.

Não obstante, durante a fase de desenvolvimento realizaram-se reuniões regulares entre as partes interessadas. Estas reuniões serviram como ponto da situação, do que tinha sido feito e o que iria ser feito até à próxima reunião, semelhante às *sprint reviews* da metodologia SCRUM [14].

Assim que foram criados os modelos, iniciou-se o desenvolvimento dos serviços web. Primeiro foram desenvolvidos os serviços de OSS; depois, o serviço do sistema de contabilidade, o módulo de *push* e, finalmente, o serviço para o *Subscriber Manager* (*pull*).

Em geral, foi seguido o caminho de um desenvolvimento ágil, sem aderir à risca a uma ou outra metodologia em particular, mas tendo o foco nos interesses da empresa e nas necessidades do projeto.

4.2 Ambientes de desenvolvimento e redundância

Ao longo do processo de implementação existiram dois ambientes de desenvolvimento que foram usados:

- *Staging* - Este é um ambiente desligado da rede real, em que não existiam consequências para os clientes caso surgissem problemas. É normalmente usado para o teste de funcionalidades, antes da passagem para um ambiente de produção. Foram usadas três máquinas: uma para servidor de políticas, uma como *Subscriber Manager* e o SCE 2000, uma versão mais antiga da plataforma.

- Produção - Só depois de estar pronto é que o sistema pode entrar neste ambiente. Após a entrada em produção, qualquer alteração realizada ao código deve passar primeiro pelo ambiente de *staging*, para ser testada. Aqui são usadas as melhores máquinas, onde se incluí o SCE 8000, e qualquer problema irá ter consequências reais. Uma outra diferença importante do ambiente de produção é que existem duas máquinas para o sistema completo e outras duas para os *Subscriber Managers*, de modo a que possa haver redundância. Se por algum motivo uma das máquinas falhar, existe outra com a mesma configuração e dados que pode ser ativa para realizar o trabalho da máquina que falhou.

Para a redundância dos dados da base de dados foi configurada e ativada a replicação do MySQL. O processo é relativamente simples: primeiro é necessário realizar um *snapshot*, uma espécie de fotografia do estado atual da base de dados da máquina mestre. Depois, importando esse *snapshot* para a outra máquina, junto com a posição dos últimos dados que foram guardados neste, podemos ativar o escravo para começar a sincronizar todas os eventos após esse *snapshot*. Para saber quais foram as operações realizadas o servidor mestre guarda uma espécie de diário, que é pedido frequentemente pelo escravo e este só tem de executar essas operações. Este mecanismo é denominado por *binary logging* [15].

4.2.1 Git - Sistema de controlo de versões

De modo a manter um histórico e ter maior controlo sobre as alterações realizadas durante a implementação, foi usado um sistema de controlo de versões distribuído denominado por Git. Distribuído significa que, se por algum motivo algum servidor que aloja o código ficar permanente indisponível, é possível restaurar o repositório de código a partir de um dos clientes [16]. Existe uma cópia completa com todas as alterações em cada cliente. É uma peça de software muito versátil que permite reverter quaisquer alterações. Para além disso torna a tarefa de distribuir as alterações realizadas, como por exemplo do ambiente de *staging* para produção, trivial.

4.3 Zend Framework

A Zend Framework, versão 1, é uma *framework* em código aberto que serve como base para o desenvolvimento de aplicações e serviços web com PHP 5. O código da *framework* é completamente orientado a objetos e cada componente é desenhado para depender o menos possível de outros componentes. Este fraco acoplamento permite a utilização de cada componente de forma individual [17].

A outra grande vantagem da *framework* é a junção de todos estes componentes, proporcionando uma robusta implementação do padrão arquitectural *Model–view–controller* (MVC), uma camada de abstração de base de dados simples de utilizar e outras bibliotecas para grande parte das tecnologias atuais.

É uma *framework* que já se encontra em utilização nos serviços internos de suporte à rede e foi um dos requisitos impostos pela empresa, sendo essa a principal razão para a sua utilização neste sistema. Permite, assim, à empresa uma maior facilidade na leitura, compreensão, manutenção do código e adição de novas funcionalidades.

4.4 Estrutura geral do projeto

Arquitecturalmente, o projecto segue uma estrutura MVC (Model, View, Controller). Contudo, no caso deste trabalho, no que toca à parte da *View* a sua utilização é bastante reduzida. Tudo o que sai do sistema são respostas em SOAP, pelo que a utilização de *layouts* e outras características mais avançadas das vistas do Zend não são necessárias. Quem realmente trata de compôr o *output* são as classes de SOAP da *framework*, instanciadas nos controladores. No fundo, a maior parte do trabalho é realizada pelos modelos. Todo o código dos serviços web corre num servidor Apache, com exceção do módulo *push*, que é o único módulo que corre sem necessidade de estar ligado a um servidor web,

usando apenas o executável do PHP. Hierarquicamente, o projeto encontra-se organizado como mostra a figura 4.1.

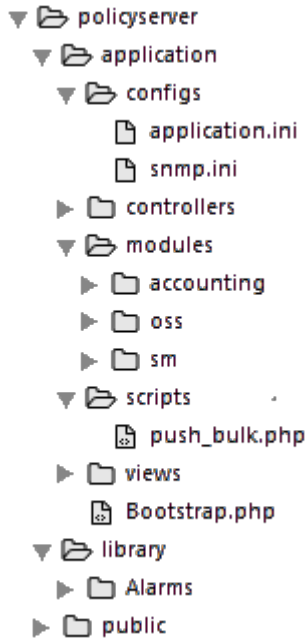

Figura 4.1: Organização e hierarquia das pastas do projeto

Tal como mostra a figura 4.1, a organização das pastas e dos ficheiros do projeto é a seguinte:

- A pasta `application` é onde está a lógica da aplicação/projeto; é onde se insere a estrutura MVC assim como configurações, módulos e scripts;
- A pasta `application/configs` contém as configurações do projeto (mais detalhes na secção 4.4.1)
- As pastas `application/controllers` e `application/views` é onde estão inseridos os controladores e vistas pai do projeto. Na prática, não servem nenhum propósito pois todo o código está separado por módulos;
- A pasta `application/modules` contém os módulos, e cada módulo uma estrutura que em tudo se assemelha à estrutura MVC da pasta `application`;
- A pasta `application/scripts` contém o modulo de *push* que é o único componente usado fora do contexto MVC e encontra-se, por isso, separado;
- A pasta `library` contém uma outra pasta chamada `Alarms`, que inclui as classes criadas para lidar com o envio de alarmes;
- Por fim, a pasta `public` contém todos os ficheiros públicos da aplicação. O único ficheiro relevante aqui é o `index.php`, pois é este que invoca a classe

`Zend_Application`, que serve essencialmente como o ficheiro de arranque para qualquer pedido direcionado ao servidor.

4.4.1 Configurações do projeto

As configurações de um projeto Zend são simples ficheiros .INI, de texto, com secções e propriedades. Cada propriedade tem um nome e um valor separados por um sinal de igual. As secções permitem agrupar as propriedades. Por exemplo:

```
[secção]
nome1=valor1
nome2=valor2
```

4.4.1.1 Da aplicação

O ficheiro de configuração de um projeto, `application.ini`, contém valores por omissão, necessários para o funcionamento deste. Contudo, é possível adicionar propriedades para qualquer propósito. Incluídas neste ficheiro estão as configurações para a base de dados, ficheiros e URL's para o sistema de contabilidade e *Subscriber Manager*.

4.4.1.2 De SNMP

De modo a poder enviar alarmes (*traps*) para um ou mais SNMP *Managers* foi criado um ficheiro extra de configuração, `snmp.ini`, onde facilmente se podem definir novos *managers*.

```
; Formato hostname,communitystring
managers[] = 10.0.0.100,public
```

4.4.2 Configuração do *Subscriber Manager*

Foi necessário também realizar a configuração do *Subscriber Manager* e do seu módulo SOAP LEG. Embora faça parte de implementação, optou-se por incluir um guia de configuração da máquina no anexo C, para que ficasse também como um documento de referência para a empresa.

Antes de passarmos aos módulos propriamente ditos, convém identificar algumas partes que são comuns a todos.

4.4.3 Constantes do projeto

Ao longo do projeto irão aparecer referências a constantes como `APPLICATION_PATH`, `APPLICATION_ENV`, entre outras. Estas constantes são particularmente úteis pois facilitam a especificação dos caminhos de diferentes ficheiros em relação ao próprio projeto ou o ambiente em que está a correr, respetivamente. Para o código que corre sobre o servidor web, estas constantes estão sempre disponíveis pois encontram-se definidas no ficheiro de *bootstrap*, `public/index.php`. Caso contrário, estão geralmente definidas no topo de qualquer *script* que use as bibliotecas da *Zend Framework*.

4.4.4 Modelos e a ligação à base de dados

Todos os modelos têm de saber como fazer a ligação à base de dados para poderem realizar as operações necessárias. Assim, inclui-se no construtor do modelo essa mesma ligação, construída a partir das configurações da aplicação, referidas na secção anterior:

```
1 function __construct()
2 {
3     $config = new Zend_Config_Ini(APPLICATION_PATH .
4         '/configs/application.ini', APPLICATION_ENV);
5     $this->config = $config;
6     $this->db = Zend_Db::factory($config->database);
7 }
```

Como consta na linha 6, a ligação à base de dados faz-se com recurso a uma fábrica que está integrada como um método estático da classe de base de dados da *framework*. Esta fábrica retorna o adaptador pronto a utilizar. Tudo o que temos de fazer é passar os parâmetros de configuração. Apesar de se encontrar dentro do construtor, este código é válido para outros componentes *stand-alone* como, por exemplo, o módulo de *push*.

4.4.5 Respostas SOAP

De modo a integrar com as outras plataformas da empresa, cada resposta dos modelos deve incluir um objecto *response* que incluí um identificador de erro e descrição desse erro. Se não ocorrerem erros então o identificador será 0 e a descrição OK. O método que se segue demonstra a construção desse objeto de resposta:

```
1 private function response($id, $text) {
2     $response = new stdClass();
3     $response->errorId = $id;
4     $response->errorText = $text;
5     return $response;
```

```
6 }
```

Se existirem dados para retornar, esses são simplesmente anexados a uma qualquer variável desse mesmo objeto.

4.5 Módulos dos serviços Web

A primeira parte da implementação passou pela criação dos serviços web, nomeadamente o módulo de OSS, permitindo também assim que houvesse uma base inicial para a introdução, visualização e gestão dos dados do sistema. Os 3 módulos que constituem esta parte do sistema permitem a comunicação e realizam as tarefas relacionadas com cada uma das entidades externas: o sistema de contabilidade, o *Subscriber Manager* e os serviços internos de informação.

4.5.1 Módulo OSS

Este módulo é o que providencia todas as operações para os serviços de informação. O número 2.(c) dos requisitos funcionais (secção 3.1.1) especifica quais são estas operações. Contudo, vamos abordar uma a uma para demonstrar como foram implementadas.

4.5.1.1 Controlador

A nível dos controladores, existe apenas um, *IndexController*. Este módulo tem apenas de realizar duas funções:

- Disponibilizar o WSDL que descreve as operações;
- Receber e responder aos pedidos dos serviços descritos no WSDL.

Disponibilização do WSDL

A geração dos WSDL's neste trabalho é realizada de forma automática, com recurso à classe `Zend_Soap_AutoDiscover`. Isto permite que o WSDL gerado tenha a mesma estrutura que é utilizada nos outros serviços já existentes, facilitando a sua integração. Esta classe, acima de tudo, simplifica a criação de um WSDL correto, de acordo com a especificação. Qualquer alteração que seja realizada numa das funções do serviço é refletida imediatamente.

Para que a auto-descoberta funcione é necessário que o programador adicione comentários antes de cada método da classe, incluindo uma descrição do método, tipos de dados dos parâmetros a receber e tipo de dados da resposta. Por exemplo:

```
/**
 * Dado um ou mais IP's...
 *
 * @param Array \$ips
 * @return object
 */
public function getPairs($ips)
...
```

Estes comentários, normalmente denominados por PHP *DocBlocks* [18], devem estar corretamente definidos pois é a única maneira da classe de auto-descoberta determinar os tipos de dados, para além de ser, também, uma boa prática.

Para a disponibilização do WSDL é adicionada, no respetivo controlador do módulo, uma nova ação:

```
1  public function wsdlAction()
2  {
3      $this->getHelper("ViewRenderer")->setNoRender(true);
4      $wsdl = new Zend_Soap_AutoDiscover(true, "http://" .
5          $_SERVER['HTTP_HOST'] . "/oss/index/soap");
6      $wsdl->setOperationBodyStyle(
7          array('use' => 'literal')
8      );
9      $wsdl->setClass("Oss_Model_Services");
10     $wsdl->handle();
11 }
```

A primeira função, na linha 3, garante que não é realizado nenhum *output* de uma vista pois só estamos interessados em respostas SOAP. Na linha 4 é instanciada a classe que usamos para a geração do WSDL. O primeiro parâmetro especifica que deve ser usada a estratégia normal para a deteção de tipos complexos. O segundo parâmetro especifica o *Uniform Resource Identifier* (URI) que deve ser utilizado para as chamadas ao serviço.

A função da linha 6 faz com que as *tags* geradas sejam mais simples, de acordo com a norma dos serviços existentes. Finalmente, é na linha 9 que são especificadas as funções a usar para a geração. Neste caso é utilizada uma classe, o modelo para este módulo. A última linha faz com que seja a classe de auto-descoberta a tratar do pedido do WSDL para o utilizador.

Resposta aos pedidos SOAP

O código das funcionalidades deste e dos outros módulos, encontra-se no modelo (secção 4.5.1.2) do padrão arquitetural MVC. Mas antes disso, da mesma forma que existe

uma acção no controlador para a disponibilização do WSDL, também terá de existir uma para os pedidos ao serviço em si:

```
1 public function soapAction()
2 {
3     $this->getHelper('ViewRenderer')->setNoRender(true);
4     $server = new Zend_Soap_Server("http://" .
5         $_SERVER['HTTP_HOST'] . "/oss/index/wsdl");
6     $server->setClass('Oss_Model_Services');
7     $server->handle();
8 }
```

Desta vez a classe em que estamos interessados é a `Zend_Soap_Server`. Como já temos um documento WSDL preparado, apenas temos de indicar a sua localização e a classe a utilizar (a mesma que foi usada na geração do WSDL). Este servidor trata de receber e interpretar os pedidos e envia a devida resposta. Passemos assim ao modelo.

4.5.1.2 Modelo

Todas as funções deste modelo têm uma estrutura semelhante. Sempre que existirem métodos, partes distintas ou novas, estas serão abordadas apenas da primeira vez que aparecem, e omitidas caso contrário.

Função getPairs

Esta primeira função permite a consulta dos *packages* dado um ou mais IPs.

```
1  public function getPairs($ips)
2  {
3      if (empty($ips)) {
4          return $this->response(0, "OK");
5      }
6
7      $query = "SELECT INET_NTOA(network_id) AS ip, subnet_mask, package_id
           FROM pairs WHERE network_id in (";
8      foreach ($ips as $ip) {
9          $query .= "INET_ATON('$ip'),";
10     }
11     $query = substr($query, 0, -1);
12     $query .= ")";
13
14     try {
15         $response = $this->response(0, "OK");
16         $result = $this->db->fetchAll($query, null, Zend_DB::FETCH_OBJ);
```

```
17          $response->pairs = $result;
18          return $response;
19      } catch(Exception $ex) {
20          return $this->response(1, $ex->getMessage());
21      }
22  }
```

A primeira condição garante que, se não forem recebidos IP's, é dada uma resposta vazia. Não é estritamente necessário, mas evita fazer-se uma tentativa de consulta que irá retornar erro. A segunda parte do código, da linha 7 à 12, trata da construção da *query* à BD.

Nesta parte é importante referir algumas coisas que ainda não foram abordadas. Nas respostas enviadas por estas funções só estamos interessados na notação dos grupos de octetos com pontos. Para realizar a conversão é utilizada uma função própria do MySQL, `INET_NTOA`, que converte da notação decimal para a notação com pontos, que é a mais natural. Ainda outro ponto é o facto de não existir limpeza do *input* fornecido. Este sistema trabalha de forma autónoma, pelo que não existe intervenção humana a nível do fluxo dos dados.

Para um pouco mais de eficiência é recomendado enviar apenas um comando de seleção à BD. Para que isso seja possível temos de construir a *query* de modo a aceitar um número indeterminado de IPs. Cada IP recebido é concatenado entre parêntesis e separado por vírgula. Estes IPs também precisam de ser convertidos na outra direção, através da função `INET_ATON`. Depois de serem todos concatenados, retira-se a última vírgula adicionada (linha 11) e fecha-se a *query* (linha 12), para que esta seja válida.

Por fim, a query é executada dentro de um bloco *try-catch*. É importante, por causa das normas nos serviços já existentes, que os dados retornados venham na forma de objectos que depois são traduzidos em *tags* literais na resposta SOAP. Para que isso aconteça, tem de ser usada a constante `FETCH_OBJ`, da classe `Zend_Db`, no terceiro parâmetro da função *fetchAll*. O segundo parâmetro é para as *bindings* da query, que por norma neste sistema nunca serão utilizadas, e está por isso a *null*.

Se tudo correr bem, é enviada a resposta dentro do objeto de retorno. Se não, então é enviado o motivo do erro junto com o identificador. A escolha do identificador não depende de nenhuma regra em específico, contudo, é importante ser consistente e no futuro não mudar o identificador para outro tipo de erro completamente diferente, pois isso poderá afetar a integração já feita.

Função getPackageIps

Esta função é a inversa da anterior. Dado um *package* retorna os endereços IP associados a este.

```
1  public function getPackageIps($package)
2  {
3      ...
4      $query = "SELECT INET_NTOA(network_id) AS ip, subnet_mask FROM pairs
           WHERE package_id = '$package'";
5      try {
6          $response = $this->response(0, "OK");
7          $response->ips = $this->db->fetchAll($query, null, Zend_DB::
               FETCH_OBJ);
8          return $response;
9      ...
10 }
```

Função getPackageCount

Esta função que se segue retorna o número de IPs num dado package. A função de `count()` do MySQL torna esta tarefa simples, basta indicar qual o *package_id* na cláusula `WHERE`. Se este não existir, irá ser retornado 0. A utilização do método `fetchOne` é uma maneira rápida de devolver numa *string* apenas o resultado da primeira coluna e linha, que é o caso.

```
1  public function getPackageCount($package)
2  {
3      ...
4      $response->count = $this->db->fetchOne("SELECT COUNT(*) as count FROM
           pairs WHERE package_id = " . $package);
5      return $response;
6      ...
7  }
```

Função getCountPerPackage

A próxima função retorna todos os identificadores de *package* existentes em conjunto com o respectivo número de IPs associado a cada um desses *packages*. A claúsula `GROUP BY` é que trata de agregar todas as entradas, pelo que basta selecionar a coluna *package_id* em conjunto com a função `count()` para que seja devolvida a contagem para cada um dos *packages*.

```
1 public function getCountPerPackage()
2 {
3     ...
4     $response->counts = $this->db->fetchAll("SELECT package_id, COUNT(*)
          as count FROM pairs GROUP BY package_id ORDER BY package_id", null,
           Zend_DB::FETCH_OBJ);
5     return $response;
6     ...
7 }
```

Função getAllOverrides

Era importante, também, poder obter-se todos *overrides*. Uma simples seleção trata deste problema.

```
1 public function getAllOverrides()
2 {
3     ...
4     $response->overrides = $this->db->fetchAll("SELECT INET_NTOA(
          network_id) AS ip, subnet_mask, package_id, date_limit FROM
          overrides", null, Zend_DB::FETCH_OBJ);
5     return $response;
6     ...
7 }
```

Função getOverrides

Dada uma lista de IP's, esta função retorna os respectivos *overrides*, se existirem. Muito semelhante à função `getPairs`:

```
1  public function getOverrides($ips)
2  {
3      ...
4      $query = "SELECT INET_NTOA(network_id) AS ip, package_id, date_limit
           FROM overrides WHERE network_id in (";
5      foreach ($ips as $ip) {
6          $query .= "INET_ATON('$ip'),";
7      }
8      $query = substr($query, 0, -1);
9      $query .= ")";
10
11     try {
12         $response = $this->response(0, "OK");
```

```
13          $response->overrides = $this->db->fetchAll($query, null, Zend_DB::
                FETCH_OBJ);
14          return $response;
15      ...
16  }
```

Função addOverride

Esta função adiciona um *override* na base de dados e envia de seguida o IP e *package* do *subscriber* para o SM, para que seja aplicado. Se o *override* já existir, então será atualizado. Por partes, primeiro temos:

```
1  public function addOverride($ip, $subnet_mask, $packageid, $date_limit)
2  {
3      ...
4      if (strtotime($date_limit) <= time()) {
5          return $this->response(2, "Date in the past.");
6      }
```

Se a data inserida for no passado de pouco vai servir. Devemos terminar e alertar o utilizador desse facto com um erro. Depois para se inserir na base de dados, é usada a seguinte expressão:

```
1  $query = "INSERT INTO overrides (network_id, subnet_mask, package_id,
       date_limit) VALUES (INET_ATON('$ip'), '$subnet_mask' , '$packageid', '
       $date_limit') ON DUPLICATE KEY UPDATE package_id = VALUES(package_id),
       date_limit = VALUES(date_limit)";
2  $this->db->query($query);
```

`ON DUPLICAYTE KEY UPDATE` permite que um registo existente seja actualizado com `VALUES`. No fundo esta *query* funciona como 2 em 1: se não existir é inserido, se existir é actualizado.

A secção de código que se segue trata de enviar o novo *package* para o *Subscriber Manager*. É instanciada a classe `Zend_Soap_Client` com o URL do serviço e mudada a versão do SOAP para 1.1, de modo a que seja compatível com o servidor de SOAP do *Subscriber Manager*. Depois, é necessário criar uma variável do tipo `SoapVar` para ser corretamente interpretada pelo serviço do SM. O primeiro parâmetro recebe os dados a enviar, o segundo especifica a codificação destes, o terceiro o tipo e o quarto o *namespace* do tipo. Enquanto o terceiro e quarto parâmetros estão de certa forma especificados no WSDL, a codificação a utilizar é essencial e tratou-se de um processo empírico.

```
1  $client = new Zend_Soap_Client(
2      $this->config->subscribermanager->url
3  );
```

```
4  $client->setSoapVersion(SOAP_1_1);
5
6  $querySubscriberIn = new SoapVar(
7      array(
8          "subscriberId" => $ip . "/" . $subnet_mask,
9          "mappings" => array($ip . "/" . $subnet_mask),
10         "keys" => array("packageId"),
11         "values" => array($packageid)
12     ),
13     XSD_ANYTYPE,
14     "QuerySubscriberIn",
15     "http://generated.soapserver.management.pcube.com"
16 );
```

Agora é necessário proceder ao envio desta nova informação. O nome da função a utilizar do lado do SM chama-se `updateSubscriberDetails` e é exatamente isso que invocamos, junto com a variável criada no passo anterior.

Contudo, para que a implementação ficasse mais robusta, esta invocação pertence a um ciclo de tentativas que só termina se for recebida uma resposta afirmativa do servidor do SM ou se se esgotarem as 4 tentativas especificadas. Se forem verificados os *headers* da resposta pelo código `HTTP/1.0 200 OK`, sabemos se foi recebido com sucesso ou não. Se o envio falhar, então teremos de avisar o utilizador com um erro. E assim termina esta função.

```
1          $sent = false;
2          $tries = 0;
3
4          $client->updateSubscriberDetails($querySubscriberIn);
5          while( ! $sent && $tries < 4 ) {
6              $client->updateSubscriberDetails($querySubscriberIn);
7              $sent = strpos($client->getLastResponseHeaders(), "HTTP/1.0
                   200 OK") !== false;
8              $tries++;
9          }
10
11         if (!$sent) {
12             $response = $this->response(3, "Failed to push override to the
                    \emph{Subscriber Manager}. Please try again in a few
                   moments.");
13         }
14
15         return $response;
16     } catch (Exception $ex) {
17         return $this->response(1, $ex->getMessage());
```

```
18      }
19  }
```

Função removeOverride

Embora os *overrides* expirem na data limite estabelecida, é conveniente ter a possibilidade de removê-los mais cedo. O primeiro passo é eliminar o *override* da base de dados:

```
1  public function removeOverride($ip, $subnet_mask)
2  {
3      ...
4      $this->db->beginTransaction();
5
6      try {
7          $response = $this->response(0, "OK");
8
9          if ($this->db->delete("overrides", "network_id = INET_ATON('$ip')
               AND subnet_mask = '$subnet_mask'") != 1) {
10             $response = $this->response(2, "That override record doesn't
                   exist.");
11             $this->db->rollBack();
12             return $response;
13         }
```

No caso de surgirem erros inesperados ao apagar o *override* optou-se por incluir as operações na base de dados dentro de uma transação, sendo facilmente revertíveis quaisquer alterações. Não está documentada, nesta versão, nenhuma operação de remoção de um subscriber através de SOAP disponibilizado pelo SM. Existem, portanto, duas opções neste momento: ou atualizar o *package* para o último conhecido ou, caso contrário, remover quaisquer IPs que estejam mapeados e atribuir o *package* 0, que é equivalente ao *package* de um subscriber que anónimo.

```
1          $pair = $this->db->fetchRow("SELECT INET_NTOA(network_id) AS
               network_id, subnet_mask, package_id FROM pairs".
2          " WHERE network_id = INET_ATON('$ip') AND subnet_mask = '
               $subnet_mask'", null, Zend_DB::FETCH_OBJ);
3
4          if ($pair != false) { // Se for encontrado um registo prévio
5              $querySubscriberIn = new SoapVar(
6                  array(
7                      "subscriberId" => $pair->network_id . "/" . $pair->
                           subnet_mask,
```

```
8                       "mappings" => array($pair->network_id . "/" . $pair->
                            subnet_mask),
9                       "keys" => array("packageId"),
10                      "values" => array($pair->package_id)
11                  ),
12                  ...
13              );
14          } else {
15              $querySubscriberIn = new SoapVar(
16                  array(
17                      "subscriberId" => $ip . "/" . $subnet_mask,
18                      "keys" => array("packageId"),
19                      "values" => array(0)
20                  ),
21                  ...
22              );
23          }
24
25          $client->updateSubscriberDetails($querySubscriberIn);
26
27          if (strpos($client->getLastResponseHeaders(), "HTTP/1.0 200 OK")
                === false) {
28              $this->db->rollBack();
29              $response = $this->response(3, "Failed to push remove override
                     to the SM.");
30          } else {
31              $this->db->commit(); // Agora que temos confirmação podemos
                    fazer commit
32          }
33          return $response;
34      } catch(Exception $ex) {
35          $this->db->rollBack();
36          return $this->response(1, "Failed to delete override. Reverted any
                 changes. Reason: ".$ex->getMessage());
37      }
38  }
```

No caso de não termos um registo prévio e o *package* ficar a 0 não é caso para alarme. Logo que existir tráfego por parte desse IP, este será atualizado normalmente e a nova informação substituirá a anterior.

Agora que todas as funções foram abordadas conclui-se, assim, o módulo dos serviços OSS. O próximo módulo trata da interoperabilidade entre o sistema de contabilidade e o nosso sistema.

4.5.2 Módulo contabilidade

Este módulo tem como responsabilidade a disponibilização de um *serviço* que permita ao sistema de contabilidade o envio de nova informação na forma de (IP, *package*). Este serviço é o ponto de entrada da informação que flui pelo resto do sistema. Após os pares serem armazenados na base de dados subsequentemente é lançado, assincronamente, o módulo externo de *push*.

4.5.2.1 Controlador e modelo

Em termos da implementação, esta é idêntica ao módulo OSS. Só diferem os caminhos (URLs) de acesso aos serviços. Existe apenas uma função no modelo: `pushBulk`.

Função pushBulk

Tipicamente, um pedido para este serviço vem no seguinte formato:

```
<soapenv:Envelope ... >
    <soapenv:Header/>
        <soapenv:Body>
        <soap:pushBulk>
            <pairs>
                <pair>
                    <ip>192.168.1.100/32</ip>
                    <package>1</package>
                </pair>
            </pairs>
        </soap:pushBulk>
    </soapenv:Body>
</soapenv:Envelope>
```

É esperado um array de um ou mais pares (IP, *package*) `<pair>`, em que cada um tem de ter os atributos `<ip>` e `<package>`.

Como se trata de uma comunicação entre dois sistemas autónomos, é conveniente que esta função retorne verdadeiro no caso do *push* dos dados ter sido realizado com sucesso e falso caso contrário.

```
public function pushBulk($pairs)
{
    if (empty($pairs)) {
        return false;
    }
```

Se não existirem pares, podemos imediatamente retornar falso. Na próxima parte é introduzido o conceito do número `serial`:

```
1     $this->db->beginTransaction();
2
3     try {
4         // Obter o último serial utilizado
5         $result = $this->db->fetchRow("SELECT id, val FROM counters WHERE
              id = 'pairs'",
6             NULL, Zend_DB::FETCH_OBJ);
7
8         $serial = $result->val + 1;
9
10        // Se o próximo número exceder o limite então o próximo serial ser
              á 0
11        if ($serial > 2147483646) {
12            $serial = 0;
13            // Atualizar os registos antigos com -1 (reset)
14            $this->db->query("UPDATE pairs SET serial = -1");
15        }
```

De modo a que o módulo exterior de *push* saiba quais são os pares a enviar para o *Subscriber Manager*, é preciso que exista uma referência. Optou-se então por guardar um número decimal na base de dados que servisse como essa referência. Isto é necessário porque queremos desacoplar o processo que realiza o *push* do processo do servidor web, para que o sistema de contabilidade não fique à espera de uma resposta.

Existe, contudo, um limite para o tipo de dados `signed INT`, que é `2147483647` [19]. Não foi usado o tipo de dados `unsigned` pelo facto de ser preciso utilizar um número negativo para indicar que foi ultrapassado esse limite. Neste caso optou-se por fazer uma verificação no limite máximo (linha 12) e se for ultrapassado atualizar a coluna do `serial` de todos os registos já existentes com o valor -1. Assim, o próximo `serial` a ser usado será o 0. Se a coluna *serial* de todos os pares não fosse atualizada quando já não há mais espaço para incrementar essa coluna (limite de um inteiro de 32 bits) registos mais antigos poderiam ser inseridos incorretamente mais tarde, porque teriam um número *serial* igual ao do que estaria a ser processado nesse momento.

```
1         $query = "INSERT INTO pairs (network_id, subnet_mask, package_id,
              serial) VALUES ";
2
3         foreach($pairs as $pair) {
4             // Se por algum motivo algum dos atributos não estiver
                  presente ignoramos e passamos à frente
5             if (empty($pair->ip) || empty($pair->package)) {
6                 continue;
7             }
```

```
8             $ip_address = substr($pair->ip, 0, -3); // Remove-se a máscara
                  CIDR para obter só o IP
9             $subnet_mask = substr($pair->ip, -2); // Inverso para obter a
                 máscara
10            $query .= "(INET_ATON('" . $ip_address . "'), '" .
                 $subnet_mask . "', '" .
11                          $pair->package . "','" . $serial . "'),";
12        }
13        $query = substr($query, 0, -1); // Retira-se a última vírgula
14        // Se o registo já existir actualiza-se apenas o package e o
             serial
15        $query .= " ON DUPLICATE KEY UPDATE package_id = VALUES(package_id
             ), serial = VALUES(serial) ";
16        $this->db->query($query);
```

Depois de se atualizarem todos os pares só temos de atualizar o `serial` que foi utilizado e terminar com o `commit`.

```
1         // Atualizar serial
2         $query = "UPDATE counters SET val=$serial WHERE id='pairs' LIMIT 1
              ";
3         $this->db->query($query);
4         // Commit das alterações
5         $this->db->commit();
```

Agora temos de chamar uma das funções que permite executar outros programas, a função `shell_exec` [20]:

```
1         $cmd = "php " . APPLICATION_PATH . "/scripts/push_bulk.php $serial
              > /dev/null 2>/dev/null &";
2         $this->logger->log("Calling: " . $cmd, Zend_Log::DEBUG);
3         shell_exec($cmd);
4     } catch (Exception $ex) {
5         // Se algum dos passos antes do commit falhou podemos fazer
             rollback de qualquer alteração
6         $this->db->rollBack();
7         $this->logger->log($ex->getMessage(), Zend_Log::EMERG);
8         return false;
9     }
10    return true; // Se tudo correr bem
11 }
```

Normalmente ao executar-se a função `shell_exec` o processo pai espera pela conclusão do processo lançado. Contudo, é possível evitar esse comportamento ao redirecionar todo o output do processo para `/dev/null`, que é um dispositivo virtual do sistema que descarta todos os dados escritos para este, e adicionar o símbolo *ampersand* (&), o

que faz com que o processo fique a correr por trás. Desta maneira podemos chamar o módulo de *push* e retornar imediatamente uma resposta ao sistema de contabilidade. A grande vantagem é o facto deste último não ter de ficar preso à espera que o processamento termine.

4.5.3 Módulo SM

A única responsabilidade deste módulo é certificar-se que responde aos pedidos de *pull* provenientes do *Subscriber Manager*. O manual da SOAP LEG [21] do *Subscriber Manager* apresenta um documento WSDL para ser utilizado na formulação das respostas aos pedidos. Este documento encontra-se também no anexo B.

A `soapAction` do controlador deste módulo difere ligeiramente das anteriores:

```
1  ...
2  public function soapAction()
3  {
4      $this->getHelper('ViewRenderer')->setNoRender(true); // Desactivar
           carregamento de views
5      $options = array(
6          'soap_version' => SOAP_1_1, // O Subscriber Manager opera na
               versao 1.1 do SOAP
7          'uri' => 'http://cisco.com/CiscoQuery'
8      );
9      $server = new Zend_Soap_Server(APPLICATION_PATH . "/wsdl/
           querysubscriber.xml", $options);
10     // Definir a classe a utilizar pelo servidor
11     $server->setClass('Sm_Model_Query');
12 ...
```

Para poder retornar um objeto esperado pelo serviço foi necessária a criação de uma classe (`QuerySubscriberOut`) que contém os atributos necessários. Estes atributos estão definidos no ficheiro WSDL mencionado acima. Para facilitar a compreensão em contexto com o código, optou-se por incluir a explicação na forma de comentários:

```
1  class QuerySubscriberOut {
2      private $subscriberId;
3      private $mappings;
4      private $propertiesKeys;
5      private $propertiesValues;
6
7      function __construct($subscriberId, $mappings = null, $propertiesKeys
           = null, $propertiesValues = null) {
8          $this->subscriberId = $subscriberId;
```

```
9           $this->mappings = $mappings;
10          $this->propertiesKeys = $propertiesKeys;
11          $this->propertiesValues = $propertiesValues;
12      }
13  }
14
15  public function QuerySubscriber($querySubscriberIn) {
16      // Guardar o IP do pedido
17      $ip = $querySubscriberIn->mappings->string;
18
19      try {
20          /**
21           * Como os IPs individuais têm precendência
22           * devemos verificá-los primeiro
23           * e só depois verificar se esse IP pertence a uma subnet.
24           **/
25          $query = "SELECT package_id FROM pairs WHERE network_id =
                INET_ATON('$ip')";
26          $result = $this->db->fetchOne($query, null, Zend_DB::FETCH_OBJ);
27
28          if ($result === false) {
29              /**
30               * Se não temos um registo /32 então vamos verificar se
                     pertence a uma subnet
31               * Estamos à procura de registos que se encaixem nesta fórmula
                     , em decimal:
32               *      máscara de rede & ip a ser verificado = ip da subnet
33               *
34               * -1 << (32 - subnet_mask) permite-nos obter a máscara de sub
                     -rede. Um bitwise shift
35               * retorna um bigint unsigned, o que significa que -1 é usado
                     com o número maior que conseguimos representar
36               */
37              $query = "SELECT INET_NTOA(network_id) AS ip, subnet_mask,
                    package_id FROM pairs " .
38                  "WHERE subnet_mask != 32 AND (-1 << (32 - subnet_mask)) &
                        INET_ATON('$ip') = network_id";
39              $subnet = $this->db->fetchRow($query, null, Zend_DB::FETCH_OBJ
                    );
40
41              if ($subnet !== false) {
42                  // Dados da subrede
43                  $data = new QuerySubscriberOut(
44                          $subnet->ip . "/" . $subnet->subnet_mask,
```

```
45                      array($subnet->ip . "/" . $subnet->subnet_mask),
46                      array('packageId'),
47                      array($subnet->package_id)
48                  );
49                  return $data;
50              } else {
51                  // Perguntar ao sistema de contabilidade se tem dados
                        sobre este IP
52                  // Aqui é usado o WSDL disponibilizado pelo sistema de
                        contabilidade
53                  $client = new Zend_Soap_Client($this->config->accounting->
                        url);
54                  // O WSDL contém um método requestPackageByIp que aceita
                        um endereço IP
55                  $result = $client->requestPackageByIp($ip);
56
57                  if (!$result) {
58                      // Não temos informação para este IP em particular
                            pelo que só especificamos um
59                      // nome para o subscriber. Por omissão este subscriber
                                terá um packageId de 0.
60                      $data = new QuerySubscriberOut(
61                              $ip . "/32"
62                      );
63                      return $data;
64                  } else {
65                      $data = new QuerySubscriberOut(
66                          $result->ip,
67                          array($result->ip),
68                          array('packageId'),
69                          array($result->package)
70                      );
71                      return $data;
72                  }
73              }
74          }
75
76          // Temos os dados para este IP/32
77          $data = new QuerySubscriberOut(
78                  $ip . "/32",
79                  array($ip . "/32"),
80                  array('packageId'),
81                  array($result)
82          );
```

```
83
84      } catch (Exception $ex) {
85          // Algo inesperado aconteceu, não iremos retornar nada
86          return new SoapVar(null, SOAP_ENC_OBJECT);
87      }
88  }
```

Nesta função nota-se a importância de guardar os IPs num formato decimal. Isso permite que seja possível realizar-se operações de *bitwise shift* e *and's* lógicos que tornam o processo de verificar se um IP se insere numa sub-rede muito mais eficiente.

4.6 Módulo de *push*

A intenção aquando a implementação deste módulo é para que fosse o mais robusto possível. Este é o módulo que realiza a atualização de todos os pares `(IP, package)` no *Subscriber Manager*, assim que os recebe do subsistema de contabilidade. Acaba por ser um dos módulos mais importantes do sistema.

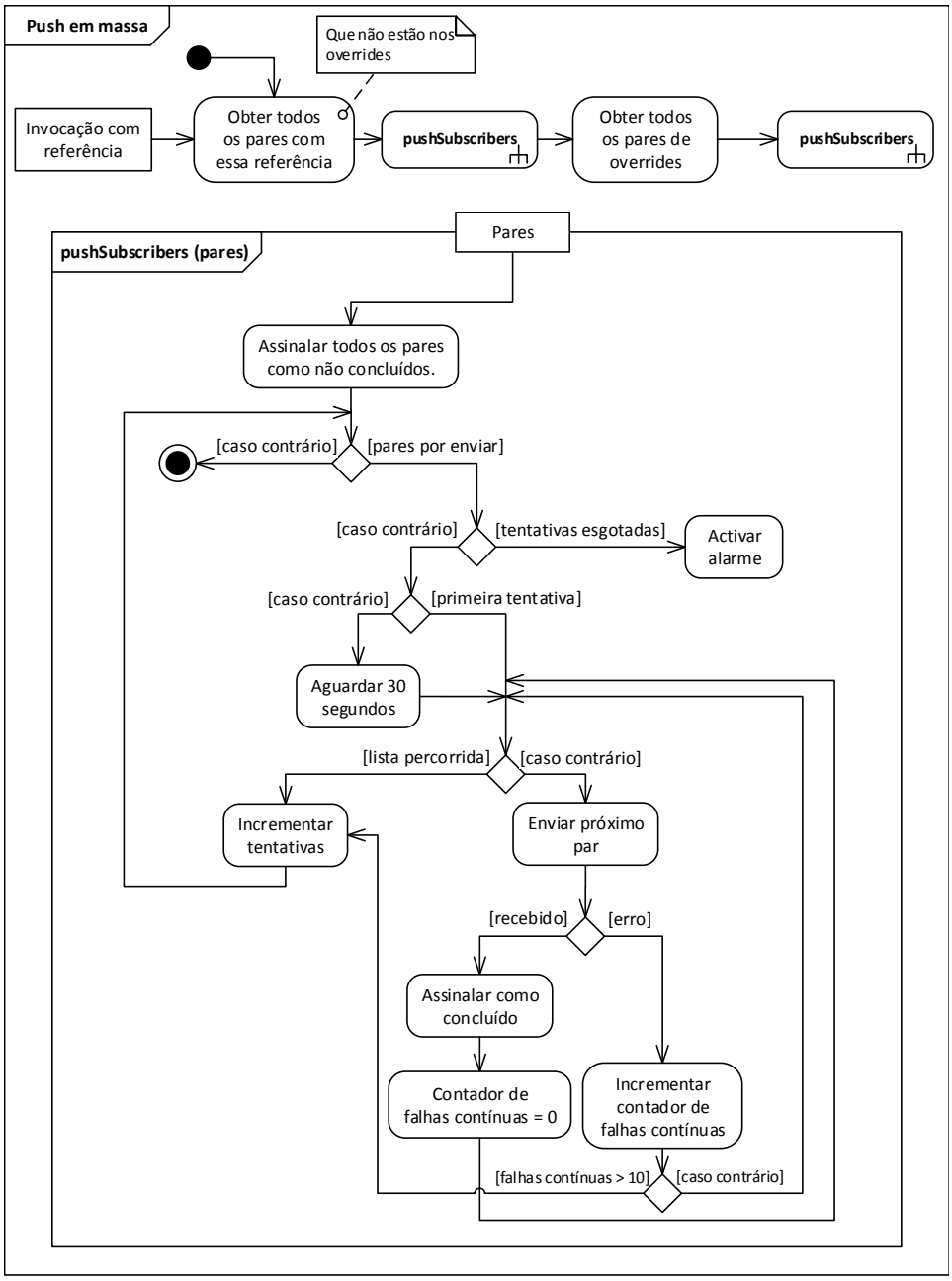

Figura 4.2: Fluxo do *push* em massa

A figura 4.2 representa o fluxo do módulo e dos dados. A parte superior do diagrama representa o fluxo global do módulo. A função `pushSubscribers` é representada na parte inferior.

Como o módulo terá de atualizar dezenas de milhares de IP's, isso representa um número igual de pedidos HTTP, pois a *SOAP Leg* só permite atualizar um *subscriber* de cada vez. Seria importante que houvesse um mecanismo de recuperação e detecção de erros, assim como o envio de alarmes, caso surgissem problemas.

Inicialmente, a invocação deste módulo tem de conter sempre uma referência para os novos pares a serem enviados. Depois existem duas fases: o *push* dos novos pares e o *push* dos overrides. É de notar que os dados a serem enviados são iguais quer para os *overrides* quer para os pares normais, pelo que pode ser utilizada a mesma função.

Função pushSubscribers

Esta função tem uma única tarefa: enviar todos os pares para o *Subscriber Manager* através de SOAP. De maneira a que falhas pontuais não invalidem o processo completamente, existe um número de tentativas (número este que pode ser alterado) do envio da lista completa. Para além disso existem também um contador de falhas contínuas de pedidos HTTP para que, por exemplo, se tivermos 40 mil pares para enviar e o SM está indisponível, não sejam feitos 40 mil pedidos que nunca vão ser concretizados. Analisando agora o diagrama da figura 4.2, o fluxo da função é o seguinte:

- No início nenhum dos pares foi enviado, pelo que ainda temos a lista completa para enviar;
- Se temos pares para enviar, continuamos; se não, a função termina aqui;
- Depois é avaliado se já foram esgotadas todas as tentativas de envio da lista:

  Se sim, significa que devemos enviar um alarme para os SNMP Managers. Se não, continuamos;

  Se esta não for a primeira tentativa devemos esperar 30 segundos no caso de existirem falhas pontuais. Caso contrário estamos na primeira tentativa e devemos continuar;
- Neste passo iteramos pela lista de pares a enviar. Se ela já tiver sido percorrida, incrementamos o número de tentativas efetuadas. Se não, então enviamos o próximo par da lista;
- Se o par enviado foi bem recebido, assinalamo-lo como concluído, fazemos *reset* do contador de falhas contínuas para 0 e passamos ao próximo par. Se houve um

erro, então incrementamos o contador de falhas contínuas e passamos ao próximo par.

Agora que já temos uma ideia de como se processa este *push* em massa, passamos então à implementação em código.

O primeiro passo é ativar a cache de WSDL's, pois o WSDL do SM é sempre o mesmo e não há necessidade de o carregar do SM em todos os envios:

```
1 ini_set('soap.wsdl_cache_enabled', '1');
```

Visto que estamos fora do contexto do servidor web, é necessário definir os caminhos e variáveis de ambiente como constantes, assim como os valores para os números de tentativas:

```
1 define('APPLICATION_PATH', realpath(dirname(__FILE__) . '/../../
      application'));
2
3 define('APPLICATION_ENV',
4 (getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') : 'development'));
5
6 define('PUSH_RETRIES', 5);
7 define('SECONDS_BETWEEN_RETRIES', 30);
```

É necessário também adicionar um caminho extra para que possamos incluir as nossas bibliotecas, onde se encontram as classes dos alarmes, de modo a que o PHP possa usar esses ficheiros:

```
1 set_include_path(get_include_path() . PATH_SEPARATOR . APPLICATION_PATH .
      "/../library");
```

Depois é só requerer as nossas dependências:

```
1 require_once "Zend/Application.php";
2 require_once "Zend/Soap/Client.php";
3 require_once "Zend/Log/Writer/Stream.php";
4 require_once "Zend/Log/Exception.php";
5 require_once "Zend/Log.php";
6 require_once "Alarms/SnmpTrap.php";
```

Criamos uma nova instância de uma aplicação Zend, muito semelhante ao código encontrado no ficheiro `public/index.php`. Inicializamos também a configuração, o adaptador para base de dados e um *logger*:

```
1 $application = new Zend_Application(
2     APPLICATION_ENV,
3     APPLICATION_PATH . '/configs/application.ini'
4 );
```

```
5 $application->bootstrap();
6 $config = new Zend_Config_Ini(APPLICATION_PATH . '/configs/application.ini
      ', APPLICATION_ENV);
7 $db = Zend_Db::factory($config->database);
8 $logger = new Zend_Log(new Zend_Log_Writer_Stream(APPLICATION_PATH . "/../
      log/push_bulk.log"));
```

De modo a que apenas uma instância deste programa se encontre a correr em qualquer momento é necessário criar uma espécie de fila de espera. A função `flock` [22] permite fazer exatamente isso, basta que para isso seja utilizado um ficheiro temporário que o sistema operativo irá tratar de libertar qualquer *lock* realizado, mesmo na ocorrência de erros:

```
1 if (! $f = fopen(APPLICATION_PATH . "/../" . $config->subscribermanager->
      push_lock_file, "w")) {
2     write_log("Can't create lock file!", Zend_Log::EMERG);
3     exit(1);
4 }
5
6 if (!flock($f, LOCK_EX)) {
7     write_log("Can't get lock!", Zend_Log::EMERG);
8     exit(1);
9 }
```

Esta restrição é imposta para que o servidor do *Subscriber Manager* não seja inundado com demasiados pedidos HTTP num curto espaço de tempo. Para além disso, poderiam haver inconsistências se, por exemplo, o processo mais antigo se atrasasse. Nessa situação, um IP atualizado primeiro pelo processo mais recente, e depois, atualizado de novo pelo processo mais antigo ficaria com um valor desatualizado.

Para evitar repetir código foi criada uma função que escreve para o ficheiro de *log*:

```
1  function write_log($message, $filter = null)
2  {
3      global $logger; // Para se poder utilizar o logger inicializado
           anteriormente
4      try {
5          if ($filter == null) {
6              if (APPLICATION_ENV != "development") return;
7              $logger->log($message, Zend_Log::DEBUG);
8          } else {
9              $logger->log($message, $filter);
10         }
11     } catch(Zend_Log_Exception $ex) {
12         echo $ex->getMessage();
13     }
```

```
14 }
```

A próxima função serve apenas para tornar o código mais legível. Recebendo um objeto com todos os pares retorna verdadeiro se todos os pares estão marcados como enviados. Falso, caso contrário:

```
1  function pairsDone($_pairs)
2  {
3      $done = true;
4      foreach ($_pairs as $i) {
5          if (!$i->done) {
6              $done = false;
7          }
8      }
9      return $done;
10 }
```

A próxima função é chamada cada vez que existe um par `(IP, package)` para ser enviado ao SM. Está, por isso, separada para poder ser reutilizada:

```
1  function updateSubscriberDetails($_client, $_pair)
2  {
3      $querySubscriberIn = new SoapVar(
4      array(
5          "subscriberId" => $_pair->ip . "/" . $_pair->subnet_mask,
6          "mappings" => array($_pair->ip . "/" . $_pair->subnet_mask),
7          "keys" => array("packageId"),
8          "values" => array($_pair->package_id)
9      ),
10     XSD_ANYTYPE,
11     "QuerySubscriberIn",
12     "http://generated.soapserver.management.pcube.com"
13     );
14
15     try {
16         $_client->updateSubscriberDetails($querySubscriberIn);
17         return strpos($_client->getLastResponseHeaders(), "HTTP/1.0 200 OK
               ") !== false;
18     } catch(Exception $ex) {
19         write_log($ex->getMessage(), Zend_Log::ERR);
20         return false;
21     }
22 }
```

A função que se segue, `pushSubscribers`, é a que foi abordada no figura 4.2. São enviados todos os pares recebidos para o *Subscriber Manager*. Termina assim que todos

os pedidos tenham sido enviados com sucesso ou se o limite de tentativas tenha sido atingido:

```
1  function pushSubscribers($_client, &$_pairs)
2  {
3      // Adicionar uma varíavel do tipo boolean nos pares que recebemos de
           modo a que posssamos manter um registo de quais é que foram
           enviados com sucesso ou não.
4      foreach ($_pairs as &$i) {
5          $i->done = false;
6      }
7
8      $attempts = 0;
9
10     while ( ! pairsDone($_pairs) ) {
11         // Em cada iteração verificamos se os dados foram bem recebidos.
12         if ($attempts < PUSH_RETRIES) {
13             // Se esta não é a primeira tentativa devemos esperar.
14             // Até lá pode ser que o servidor já esteja disponível.
15             if ($attempts > 0) {
16                 sleep(SECONDS_BETWEEN_RETRIES);
17                 write_log("Retries: " . $attempts, Zend_Log::INFO);
18             }
19
20             /**
21             * Enviar cada par
22             *
23             * Se até 5 pedidos, de seguida, falharam, significa que o
                   servidor provavelmente não está disponível. Sendo assim, sa
                   ímos desta tentativa e esperamos pelo tempo da próxima
                   tentativa. A ideia é evitar abrir ligações desnecessárias
                   quando temos, por exemplo, 40 mil pares por enviar
24             **/
25
26             $failed_in_a_row = 0; // Variável para contar o número de
                   falhas contínuas
27
28             foreach ($_pairs as &$pair) {
29                 // Se o par já foi enviado saltamos para o próximo
30                 if ($pair->done) continue;
31
32                 if (updateSubscriberDetails($_client, $pair)) {
33                     $pair->done = true; // Assinalar quen foi bem recebido
34                     $failed_in_a_row = 0; // Fazer reset ao contador
35                 } else {
```

```
36                      $failed_in_a_row++;
37                      if ($failed_in_a_row > 5) break; // Cancelar esta
                            tentativa, esperar pelo próxima
38                  }
39              }
40              $attempts++;
41          } else {
42              // Ficámos sem tentativas, devemos enviar uma notificação de
                    alarme.
43              $trap = new SnmpTrap();
44              $trap->reportBulkPushFailed();
45              return;
46          }
47      }
48  }
49  ...
50  // Criação de um novo cliente SOAP para poder chamar o serviço do SM
51  $client = new Zend_Soap_Client($config->subscribermanager->url);
52  // Definir a versão para 1.1, de modo a manter compatibilidade com a
        especificação do SM
53  $client->setSoapVersion(SOAP_1_1);
54
55  try {
56      /**
57      * Passo 1 - Recolher todos os novos pares e enviá-los para o SM
58      * Desta lista excluímos os overrides temporários
59      */
60      $query = "SELECT INET_NTOA(network_id) AS ip, subnet_mask, package_id
            FROM pairs";
61      $query .= " WHERE serial='".$serial."' AND network_id NOT IN";
62      $query .= " ( SELECT network_id FROM overrides WHERE date_limit > NOW
            () )";
63
64      $new_pairs = $db->fetchAll($query, null, Zend_DB::FETCH_OBJ);
65
66      write_log("Update, serial = $serial, " . count($new_pairs) . " new IPs
            .", Zend_Log::INFO);
67      pushSubscribers($client, $new_pairs);
68      write_log("Done updating, serial = $serial", Zend_Log::INFO);
69
70      /**
71      * Passo 2 - Agora que todos os pares foram enviados temos de aplicar
            todos os overrides temporários que ainda são válidos.
72      */
```

```
73
74      // Recolher todos os overrides ativos
75      $overrides = $db->fetchAll("SELECT INET_NTOA(network_id) AS ip,
            subnet_mask, package_id FROM overrides WHERE date_limit > NOW()",
            null, Zend_DB::FETCH_OBJ);
76
77      if (count($overrides) != 0) {
78          // Se existem overrides vamos enviá-los ao SM
79          write_log("Pushing " . count($overrides) . " overrides ...",
                Zend_Log::INFO);
80          pushSubscribers($client, $overrides);
81          write_log("Done pushing overrides.", Zend_Log::INFO);
82      }
83  } catch (Exception $ex) {
84      write_log($ex->getMessage(), Zend_Log::EMERG);
85      flock($f, LOCK_UN); // Libertar o ficheiro de lock
86  }
87
88  flock($f, LOCK_UN); // Libertar o ficheiro de lock
```

Fica assim completo o processo de envio para o *Subscriber Manager*.

4.7 Alarmes

Antes de terminar este capítulo é necessário abordar a implementação para o envio de alarmes.

De modo a manter a forma como são transmitidos os alarmes o mais abstrata possível, foi criada uma classe em que se incluem os métodos de todos os alarmes:

```
1  abstract class AbstractAlarm {
2      abstract public function reportBulkPushFailed();
3  }
```

Qualquer classe que herde desta classe abstrata terá necessariamente de implementar esses métodos, garantindo assim que todos os alarmes estão implementados, quer seja através de *traps* ou simplesmente através da escrita num ficheiro de texto.

Até à data da conclusão deste trabalho o único alarme que se revelou útil para a empresa foi o envio de alarmes quando se esgotam todas as tentativas do envio dos pares a serem processados, no módulo de *push*. No entanto, a adição de novos alarmes no futuro é bastante simples.

Em relação à implementação do envio de *traps* para o gestor SNMP, foi realizada da seguinte forma:

```
1  require_once "AbstractAlarm.php"; // Requeremos a classe abstrata
2  require_once "Zend/Config/Ini.php";
3
4  class SnmpTrap extends AbstractAlarm  {
5      private $managers;
6
7      public function __construct() {
8          $config = new Zend_Config_Ini( realpath(dirname(__FILE__) . '
               /../../application/configs') . '/snmp.ini' );
9          // Esta implementação suporta o envio para vários gestores SNMP
10         $this->managers = $config->managers->toArray();
11     }
12
13     private function sendTrap($trap) {
14         foreach($this->managers as $manager) { // Para cada um dos
               gestores
15             $args = explode(',', $manager); // Separar hostname da
                   community string
16             exec("/usr/bin/snmptrap -v 2c -c ". $args[1] ." $args[0] \"\"
                   .1.3.6.1.4.1.27394.100 " . $trap);
17         }
18     }
19
20     public function reportBulkPushFailed()
21     {
22         $this->sendTrap(".1.3.6.1.4.1.27394.100.1 s \"Failed to push pairs
                to the SM.\"");
23     }
24 }
```

O envio das *traps* depende de um programa chamado `snmptrap` que tem de estar presente no servidor.

A função `sendTrap` permite facilitar o envio de vários tipos de traps. O primeiro argumento `-v 2c` especifica que deve ser usada a versão 2c de uma notificação SNMP. O segundo argumento `-c` é onde deve ser inserida a *SNMP Community String*, uma espécie de palavra-chave que é definida no gestor de modo a que apenas notificações com essa chave possam ser aceites. O terceiro parâmetro refere-se ao endereço IP ou nome da máquina para onde se quer enviar a notificação. O quarto parâmetro, `.1.3.6.1.4.1.27394.100`, é o enterprise-OID a ser usado, especificado pela empresa. Por fim o último argumento consiste nos 3 parâmetros referentes à notificação: `OID tipo valor`. No caso da nossa notificação, enviamos um OID que é muito semelhante ao enterprise-OID, mas onde se inclui um número no final, específico à *trap*. Este valor foi também acordado.

Depois, especifica-se o tipo *string* (s) e o valor da *string*, que é um descritivo do alarme.

4.8 Conclusão

Neste capítulo foi abordada a metodologia seguida, os ambientes de desenvolvimento, redundância, ferramentas, estrutura do projeto e implementação de cada módulo com recurso às funcionalidades e bibliotecas da Zend Framework. No próximo capítulo serão apresentados os testes e os resultados obtidos.

Capítulo 5

Testes e resultados

Neste capítulo são abordados os testes realizados e os resultados conseguidos. Ao longo do processo de desenvolvimento foram realizados testes funcionais, de desempenho, fiabilidade e aceitação. Para além dos resultados obtidos nos testes é apresentado o resultado da entrada do sistema em produção.

5.1 Testes funcionais

Os dois principais tipos de testes realizados foram os *smoke tests* e os *white-box tests* para as partes mais importantes, como otimização e fiabilidade do sistema.

Os *smoke tests* são uma forma de realizar verificações para falhas rotineiras como, por exemplo, se um dado serviço web está a retornar corretamente todos os campos ou se um processo externo está a ser lançado. Se forem encontrados erros, essa parte do software não é disponibilizada até que seja corrigida [23].

White-box testing é um método para testar a estrutura interna de uma aplicação. Requer mais tempo e quem testa necessita de conhecer o código fonte a fundo. O objetivo é examinar as partes mais frágeis do código para que no final se tenha um ambiente livre de erros [24].

Não foram realizados testes automatizados, pois eram de utilidade reduzida e implicavam alocar um período de tempo durante o desenvolvimento para a sua elaboração.

5.2 Testes de desempenho

No que concerne o requisito não-funcional nº. 3, os testes de desempenho indicaram que o tempo de resposta médio para pedidos sem grandes volumes de dados anda na casa das dezenas ou poucas centenas de milissegundos, indo de encontro às expectativas. Naturalmente, para grandes volumes de dados (como por exemplo, questionar sobre os *packages* de 30 mil endereços IP's), o tempo de espera tende a aumentar, mas sempre para valores dentro do esperado.

Devido ao grande número de pedidos SOAP realizados no módulo de *push* foi importante desde o início realizar testes de desempenho, no sentido de averiguar se era de facto uma solução viável ou não. Os testes estiveram sempre abaixo dos 5 minutos durante o desenvolvimento. De modo a averiguar se era possível melhorar os tempos de envio de todos os pares para o SM, foi implementada e testada uma solução em que existiam dois processos a enviar pedidos HTTP para o servidor de SOAP do SM. Contudo, embora se poupasse alguns segundos, ocorriam falhas esporádicas do lado do SM, invalidando a solução.

5.3 Testes de fiabilidade

O sistema ficou em funcionamento permanente durante as várias semanas antecedentes à data de entrada em produção. Através do registo de eventos (*logging*) foi possível apontar quaisquer problemas que ocorressem.

O único problema durante esta fase foi a ocorrência de alguns erros durante à obtenção do documento WSDL usado para enviar informação dos *subscribers* ao SM. Por vezes essa obtenção falhava, o que fazia com que fosse ativada a próxima tentativa no ciclo do programa, ou seja, resultando numa espera de 30 segundos. Para resolver este problema ativou-se a cache de WSDL do PHP (secção 4.6), para que não fosse carregado esse documento sempre que fosse necessário atualizar os *subscribers*.

5.4 Testes de aceitação

Os testes de aceitação são realizados pelo cliente. Este tipo de testes são realizados antes de se proceder à entrega do sistema, para que o cliente os possa realizar no seu ambiente próprio de execução, e decidir se aceita, ou não, a solução [25]. Neste trabalho, os testes consistiram em verificar que todos os serviços web estavam de acordo com os requisitos estipulados para a integração com a plataforma existente (OSS). Os módulos

de *push* e do SM (*pull*) também foram alvo de uma fase de aceitação, incluindo antes e depois da entrada em produção.

5.5 Entrada em produção

Nas semanas anteriores à entrada em produção os três subsistemas já estavam a realizar todo o trabalho de recolha e cálculo em máquinas de *staging*, para que houvesse dados reais do consumo realizado pelos clientes aquando o lançamento.

A entrada em produção do Servidor de Políticas correu sem problemas. Após a mudança para o novo sistema, rapidamente se verificaram vários pedidos *pull* para aqueles IP's que ainda não tinham um *package* associado.

O tempo médio de envio dos pares para o *Subscriber Manager* rondava 2 minutos para, aproximadamente, 45 mil endereços IP. O objetivo era manter o tempo de processamento inferior a 15 minutos, que era o tempo de intervalo entre as recolhas de tráfego do servidor de Netflow.

Como se pode observar na figura 5.1, o número total de *subscribers* no sistema antigo era de, aproximadamente, 42,5 mil.

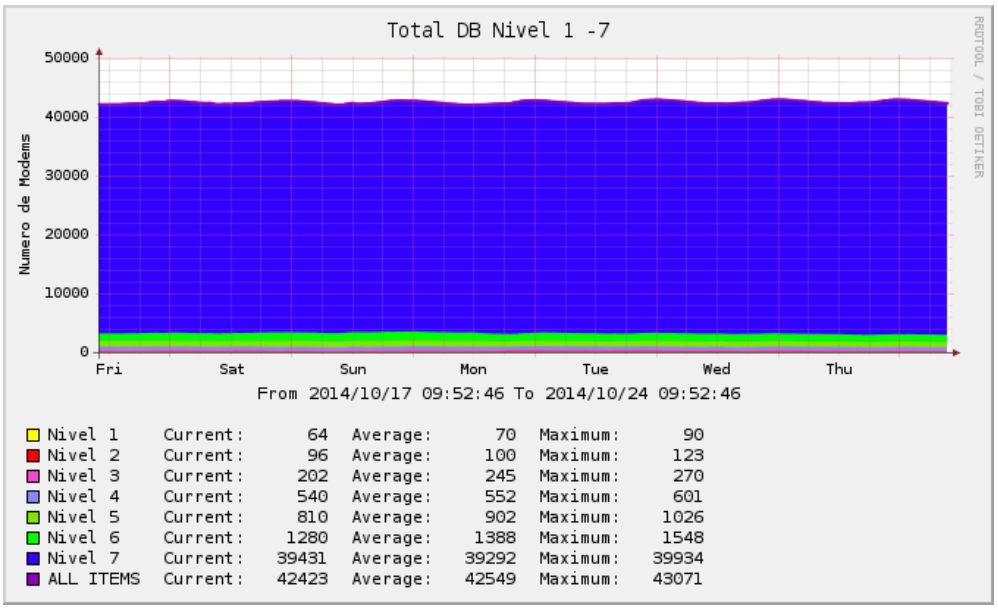

Figura 5.1: Gráfico com o número de *subscribers* no total e por *package* do sistema antigo

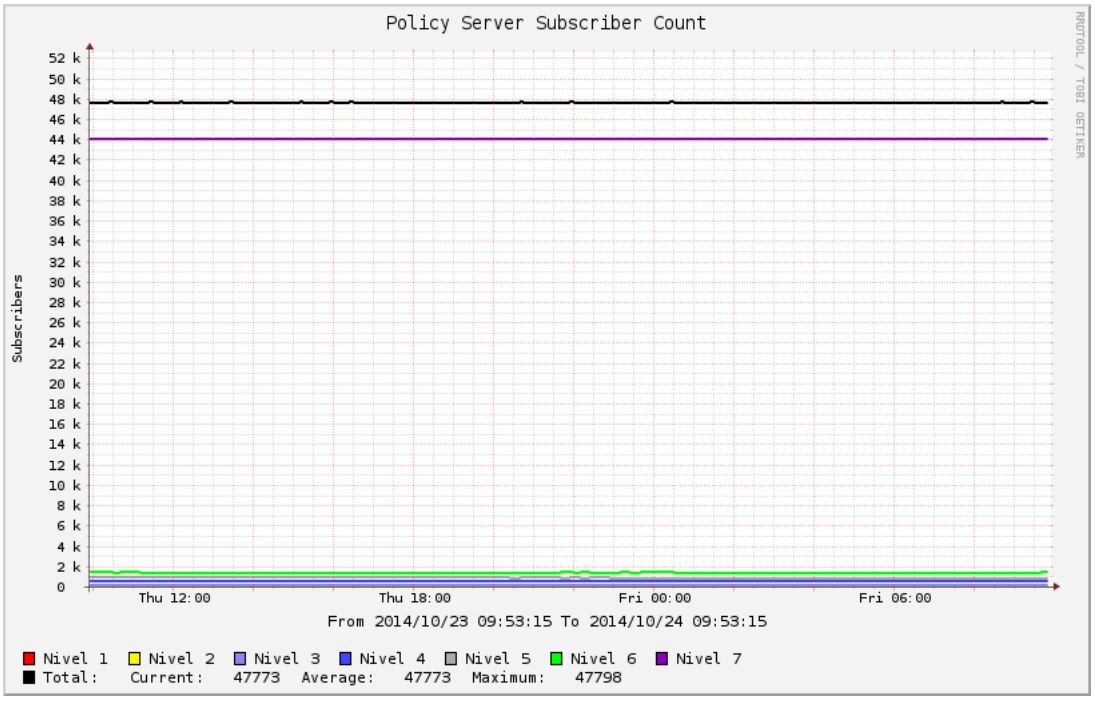

Figura 5.2: Gráfico com o número de *subscribers* no total e por *package* do novo sistema

Após a entrada em produção do novo sistema (figura 5.2) foram categorizados muito mais *subscribers* – cerca de 47,7 mil. Embora não seja inteiramente visível nestes gráficos (disponibilizados pela empresa), existiu também um ligeiro aumento do número de *subscribers* nos níveis mais baixos. No próximo capítulo é discutida a razão pela qual isto acontece (secção 6.2)

5.6 Conclusão

Neste capítulo foram mencionados os testes realizados e os resultados obtidos. No próximo capítulo são retiradas as conclusões, assim como propostas de trabalho futuro.

Capítulo 6

Conclusões e trabalho futuro

Neste capítulo é apresentado um resumo do trabalho realizado, tendo em conta os requisitos propostos, os problemas encontrados, conclusões e também sugestões de ideias ou melhorias que pudessem ser incluídas numa versão futura do sistema.

6.1 Resumo do trabalho

Este trabalho consistiu na criação de um subsistema controlador que pudesse receber pedidos para enviar, e também atualizar, os níveis de prioridade (*package*) dos assinantes (*subscribers*) na plataforma SCE. Ao mesmo tempo, foram criados também serviços web para que pudessem ser integrados na plataforma de gestão operacional da rede existente, incluindo-se aqui também a capacidade de poder substituir (*override*) temporariamente um nível de prioridade para determinado cliente, de forma simples.

Todos os objetivos delineados foram bem conseguidos quer a nível funcional, quer ao nível não-funcional. O sistema encontra-se em funcionamento autonomamente e sem problemas.

No anexo A encontra-se um diagrama de Gantt que demonstra o período de tempo alocado para cada fase do projeto.

6.2 Vantagens em relação ao sistema predecessor

No sistema antigo, os *subscribers* que ainda não tinham *package* teriam de esperar até uma hora para que lhes fosse atribuído um. Com o novo sistema, se o *package* se encontrar na base de dados do Servidor de Políticas ou do *Subscriber Manager*, o SCE recebe essa informação quase instantaneamente.

Devido à menor periodicidade, e também duração do processamento, um cliente que realize mais tráfego do que o habitual irá sentir os efeitos desse consumo mais cedo. O mesmo é válido para o inverso – volta a um nível superior mais depressa assim que deixar de realizar tanto tráfego. Isto, claro, depende muito das regras que estão em vigor, mas, no geral, acaba por ser um sistema mais justo.

Passaram, também, a existir menos *subscribers* no *package* por omissão, porque não estavam bem catalogados. Este novo sistema permitiu também que fosse mais fácil identificar os endereços IP em falta. Isto porque, qualquer endereço de IP que não esteja na base de dados estará com o *package* por omissão. Em conjunto com a nova forma (do subsistema de contabilidade) de processar o tráfego, fez com que a empresa pudesse mais facilmente identificar esses casos e corrigi-los. É possível observar esta situação nos gráficos dos resultados do capítulo anterior (secção 5.5), onde aparecem milhares de novos *subscribers*.

Embora não fosse possível, concretamente, quantificar melhorias a nível da largura de banda (por causa de diversos fatores, como por exemplo, novos clientes) esta nova flexibilidade do sistema ajuda em muito a gestão da largura de banda disponível.

Finalmente, embora seja uma característica do subsistema de contabilidade, no sistema antigo havia a restrição do conjunto de regras ser o mesmo para qualquer produto que o cliente tivesse. Isso acaba com o novo sistema.

6.3 Limitações

Devido à natureza da recolha do sistema de mapeamento e dos tempos de processamento é possível que um cliente que se ligue e adquira um IP que se encontrava em utilização por outro cliente anteriormente, fique com o *package* errado, no pior caso, durante os 30 primeiros minutos de atividade. Isto acontece porque o registo anterior para esse IP pode encontrar-se na base de dados e será utilizado até que haja uma atualização por parte do subsistema de contabilidade.

A outra limitação é o tempo de envio de todos os pares. Este processo é demorado pelo facto do servidor SOAP do SM apenas ter a capacidade de atualizar um *subscriber* por cada pedido HTTP. Ou seja, para os 47 mil *subscribers* a serem atualizados a cada 15 minutos é necessário realizar 47 mil pedidos ao servidor SOAP.

Todavia, estas são limitações que já eram conhecidas e aceites, à partida, pela NOS Madeira.

6.4 Trabalho futuro

Nesta secção apresentam-se propostas de trabalho que poderiam ser abordadas no futuro, de modo a melhorar ou a extender o sistema implementado, mas que não foram implementadas pela falta de tempo suficiente para que fossem estudadas e testadas.

6.4.1 Diferença no envio

Ao invés de serem enviados todos os pares `(IP, package)` de cada vez que existe uma atualização, poderia ser calculada a diferença dos pares entre o último envio e o envio a ser processado. Isto significaria tempos mais curtos de atualização, pois muito poucos endereços IP é que realmente mudam o seu nível de prioridade quando o intervalo de atualização se situa nos 15 minutos.

Contudo, esta solução só por si implica um fluxo normal de funcionamento, o que nem sempre acontece. Em caso de falhas ou avarias, teria de haver maneira de saber quais foram os últimos pares recebidos.

6.4.2 Catalogação *on the fly*

Numa versão futura poderia ser implementada a catalogação de acordo com a ocupação dos circuitos. O objetivo seria dar mais largura de banda quando o circuito está mais livre e evitar congestionamentos quando está mais ocupado.

Nesta implementação o Servidor de Políticas teria visibilidade sobre a ocupação dos circuitos e alteraria em tempo real os mapeamentos (IP, *package*). Por exemplo: se um determinado *subscriber* tem o *package* X, que corresponde ao nível 7 de prioridade para o seu produto, e começa a haver congestionamento então passa a ter o *package* Y. Um cliente com o mesmo produto e prioridade 6 teria o *package* Y-1. Os mapeamentos seriam sempre de acordo com a catalogação recebida do subsistema de contabilidade.

6.5 Considerações finais

A utilização do *Subcriber Manager* restringiu, de certa forma, o que é possível fazer-se com o SCE, pois utiliza software proprietário. O controlo que se tem é aquele que a documentação e o software da marca oferecem. Esta foi a principal dificuldade encontrada. Teve de haver um processo iterativo, e também de aprendizagem, para que fosse possível criar uma ponte fiável entre o Servidor de Políticas e o *Subcriber Manager* com os recursos e tempo que havia disponível.

Contudo, este trabalho foi de grande valor para a NOS Madeira, pois havia necessidade de renovar esta parte essencial do negócio, permitindo manter a competitividade e fazer a gestão da largura banda disponível com maior rapidez e facilidade.

Referências

[1] Inc. Cisco Systems. *Cisco Visual Networking Index: Forecast and Methodology, 2013–2018*. 2014. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html (acedido em 20/10/2014).

[2] D. Aguiar. «IP Network Usage Accounting - Parte I». A ser publicada. Tese de Mestrado. CCCEE, Universidade da Madeira, Funchal, Portugal.

[3] L. Ferreira. «IP Network Usage Accounting - Parte II». A ser publicada. Tese de Mestrado. CCCEE, Universidade da Madeira, Funchal, Portugal.

[4] Inc. Cisco Systems. *Cisco SCE 8000 Service Control Engine*. 2010. URL: http://www.cisco.com/c/en/us/products/collateral/service-exchange/sce-8000-series-service-control-engine/data_sheet_c78-492987.pdf (acedido em 20/09/2014).

[5] Inc. Cisco Systems. *Cisco Service Control Management Suite Subscriber Manager User Guide, Release 3.7.x*. URL: http://www.cisco.com/c/en/us/td/docs/cable/serv_exch/serv_control/broadband_app/rel37x/smug/smug.pdf (acedido em 21/09/2014).

[6] W3C. *Web Services Architecture*. URL: http://www.w3.org/TR/ws-arch/ (acedido em 22/09/2014).

[7] W3C. *Simple Object Access Protocol (SOAP) 1.1*. URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (acedido em 07/08/2014).

[8] W3C. *Web Service Definition Language (WSDL)*. URL: http://www.w3.org/TR/wsdl (acedido em 08/09/2014).

[9] Steve Burbeck. *How to use Model-View-Controller (MVC)*. 1992. URL: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html (acedido em 21/10/2014).

[10] Ian Sommerville. *Software Engineering (9th Edition)*. Addison Wesley, 2010. ISBN: 978-0-13-703515-1.

[11] James Rumbaugh, Ivar Jacobson e Grady Booch. *The Unified Modeling Language User Guide*. Second Edition. 2005. ISBN: 978-0321267979.

[12] Oracle Corporation. *MySQL 5.5 Reference Manual :: 12.16 Miscellaneous Functions*. URL: http://dev.mysql.com/doc/refman/5.5/en/miscellaneous-functions.html\#function_inet-aton (acedido em 06/09/2014).

[13] Stephen R. Palmer e John M. Felsing. *A Practical Guide to Feature-Driven Development*. Prentice Hall, 2002. ISBN: 978-0130676153.

[14] Ken Schwaber e Jeff Sutherland. *The Scrum Guide* ™. 2013.

[15] Oracle Corporation. *MySQL 5.5 Reference Manual :: 17.2 Replication Implementation*. URL: `http://dev.mysql.com/doc/refman/5.5/en/replication-implementation.html` (acedido em 26/10/2014).

[16] Scott Chacon. *Pro Git*. 2009. ISBN: 978-1430218333.

[17] Zend Technologies. *Overview - Introduction to Zend Framework*. URL: `http://framework.zend.com/manual/1.12/en/introduction.overview.html` (acedido em 06/08/2014).

[18] phpDocumentor. *phpDocumentor - Basic Syntax*. URL: `http://www.phpdoc.org/docs/latest/references/phpdoc/basic-syntax.html` (acedido em 22/09/2014).

[19] Oracle Corporation. *MySQL 5.5 Reference Manual :: 11.2.1 Integer Types*. URL: `http://dev.mysql.com/doc/refman/5.5/en/integer-types.html` (acedido em 14/09/2014).

[20] The PHP Group. *PHP: shell_exec - Manual*. URL: `http://php.net/manual/en/function.shell-exec.php` (acedido em 14/09/2014).

[21] Inc. Cisco Systems. *Cisco SCMS Subscriber Manager LEGs User Guide*. 2012. URL: `http://www.cisco.com/c/en/us/td/docs/cable/serv_exch/serv_control/broadband_app/rel37x/sm_legs/sm_legs.pdf`.

[22] The PHP Group. *PHP: flock - Manual*. URL: `http://php.net/manual/en/function.flock.php` (acedido em 22/10/2014).

[23] Steve McConnell. *Rapid Development*. 1st Edition. Microsoft Press, 1996. ISBN: 978-1556159008.

[24] Laurie Williams. *White-Box Testing*. 2006. URL: `http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf` (acedido em 20/10/2014).

[25] Brian Hambling e Pauline van Goethem. *User Acceptance Testing: A Step-By-Step Guide*. BCS, 2013. ISBN: 978-1780171678.

Anexo A

Fases do projeto

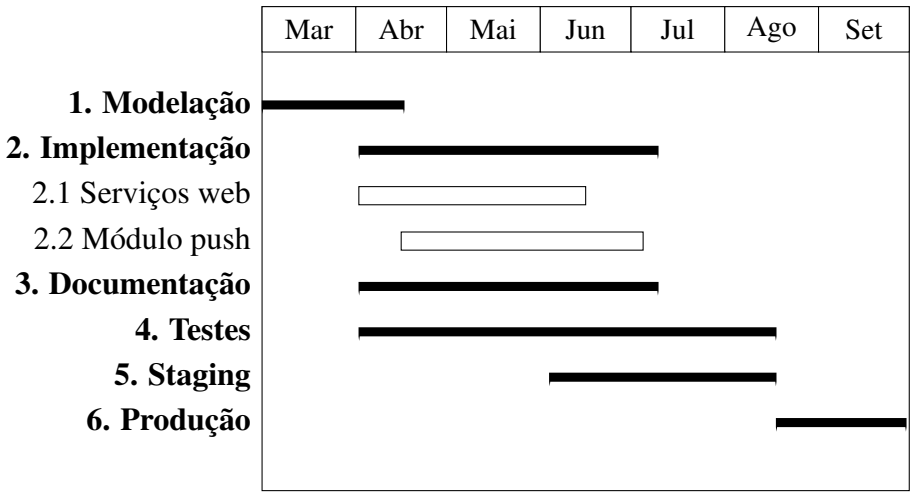

Figura A.1: Diagrama Gantt com as diversas fases do projeto

Anexo B

Cisco WSDL

```
1  <?xml version="1.0" encoding="utf-8"?>
2   <wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
3            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4            xmlns:s="http://www.w3.org/2001/XMLSchema"
5            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
6            xmlns:tns="http://cisco.com/CiscoQuery"
7            xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
8            xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
9            targetNamespace="http://cisco.com/CiscoQuery"
10           xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
11     <wsdl:types>
12       <s:schema elementFormDefault="qualified"
13       targetNamespace="http://cisco.com/CiscoQuery">
14
15         <s:complexType name="ArrayOfString">
16           <s:sequence>
17             <s:element minOccurs="0" maxOccurs="unbounded" name="string"
                    nillable="true"
18  type="s:string" />
19           </s:sequence>
20         </s:complexType>
21         <s:element name="QuerySubscriberIn">
22           <s:complexType>
23             <s:sequence>
24               <s:element minOccurs="0" maxOccurs="1" name="subscriberId"
                      type="s:string"/>
25               <s:element minOccurs="0" maxOccurs="1" name="mappings"
26              type="tns:ArrayOfString"/>
27               <s:element minOccurs="0" maxOccurs="1" name="keys" type="
                    tns:ArrayOfString"/>
```

```
28              <s:element minOccurs="0" maxOccurs="1" name="values"
29             type="tns:ArrayOfString"/>
30            </s:sequence>
31          </s:complexType>
32        </s:element>
33        <s:element name="QuerySubscriberOut">
34          <s:complexType>
35            <s:sequence>
36              <s:element minOccurs="0" maxOccurs="1" name="subscriberId"
                     type="s:string"/>
37              <s:element minOccurs="0" maxOccurs="1" name="mappings"
38             type="tns:ArrayOfString"/>
39              <s:element minOccurs="0" maxOccurs="1" name="
                    propertiesKeys"
40             type="tns:ArrayOfString"/>
41              <s:element minOccurs="0" maxOccurs="1" name="
                    propertiesValues"
42             type="tns:ArrayOfString"/>
43              <s:element minOccurs="0" maxOccurs="1" name="keys" type="
                    tns:ArrayOfString"/>
44              <s:element minOccurs="0" maxOccurs="1" name="values"
45             type="tns:ArrayOfString"/>
46            </s:sequence>
47          </s:complexType>
48        </s:element>
49      </s:schema>
50    </wsdl:types>
51    <wsdl:message name="QuerySubscriberSoapIn">
52      <wsdl:part name="parameters" element="tns:QuerySubscriberIn" />
53    </wsdl:message>
54    <wsdl:message name="QuerySubscriberSoapOut">
55      <wsdl:part name="parameters" element="tns:QuerySubscriberOut" />
56    </wsdl:message>
57    <wsdl:portType name="QueryServiceSoap">
58      <wsdl:operation name="QuerySubscriber">
59        <wsdl:input message="tns:QuerySubscriberSoapIn" />
60        <wsdl:output message="tns:QuerySubscriberSoapOut" />
61      </wsdl:operation>
62    </wsdl:portType>
63    <wsdl:binding name="QueryServiceSoap" type="tns:QueryServiceSoap">
64      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
          style="document" />
65      <wsdl:operation name="QuerySubscriber">
```

```
66        <soap:operation soapAction="http://cisco.com/CiscoQuery/
              QuerySubscriber"
67          style="document" />
68        <wsdl:input>
69          <soap:body use="literal" />
70        </wsdl:input>
71        <wsdl:output>
72          <soap:body use="literal" />
73        </wsdl:output>
74      </wsdl:operation>
75    </wsdl:binding>
76    <wsdl:service name="QueryService">
77      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Queries
            subscribers
78     data</documentation>
79      <wsdl:port name="QueryServiceSoap" binding="tns:QueryServiceSoap">
80        <soap:address location="http://localhost:8080/axis/services/
              QueryServiceSoap" />
81      </wsdl:port>
82    </wsdl:service>
83  </wsdl:definitions>
```

Anexo C

Configuração do Subscriber Manager, versão 3.7.x

Este guia fornece instruções sobre como configurar o Subscriber Manager para funcionar de acordo com os requisitos do projeto do servidor de políticas.

Requisitos e notas

A instalação do Subscriber Manager pode ser feita de forma normal, de acordo com o próprio manual [5]. O projecto foi implementado usando a distribuição de *Linux, CentOS*. Se, por algum motivo, o módulo SELinux estiver ativado no sistema operativo em modo *enforcing*, é aconselhável desactivá-lo, ou criar-se regras para que os binários do SM funcionem sem dificuldades. Para desactivar o SELinux em *CentOS*:

- No ficheiro `/etc/selinux/config`, modificar `SELINUX=enforcing` para `SELINUX=disabled`.

Por omissão, os ficheiros de configuração das próximas secções encontram-se no diretório `/opt/pcube/sm/server/root/config/`. Para conviniência de uma configuração rápida incluem-se comandos em *Bash* no final do guia.

Configuração geral - p3sm.cfg

Enumeram-se todos as secções que precisam de ser alteradas neste ficheiro. Para cada um dos parâmetros a alterar junta-se uma breve explicação.

- `[SM General]`

  `introduction_mode=pull`

  O sistema funciona baseado em modo *pull*. Se há um novo IP a fazer tráfego o SCE informa o SM. Este depois trata de enviar a informação necessária, como o packageId de volta ao SCE. Para isso, e junto com a SOAP Leg, o SM poderá pedir essa informação ao servidor de políticas.

- `[Inactive Subscriber Removal]`

  `start=yes`

  Para manter a base de dados do SM mais pequena e limpa é benéfico ativar a remoção automática de *subscribers* sem nenhum mapping (IP) ao fim de algum tempo. Este tipo de situações acontece, portanto é recomendável. Certifique-se de que o comentário # não está presente nesta linha, pois este vem por defeito.

- `[Data Repository]`

  `support_ip_ranges=yes`

  `max_range_size=262144`

  O primeiro parâmetro permite utilizar *subnets* nos *mappings*, essencial para a lógica do negócio. O segundo parâmetro determina o intervalo máximo que é usado pelo primeiro parâmetro, igualmente essencial.

- Por fim, é necessário adicionar uma secção com o IP e porta do SCE que o SM deverá comunicar e sincronizar com. Como exemplo:

  ```
  [SCE.SCE2000]
  ip=10.0.0.1
  port=14374
  ```

Configuração da SOAP LEG - soap_leg.cfg

- `start=yes`

  Este parâmetro indica que a LEG deve ser inicializada no arranque.

- `supported_pull_type=anonymous`

  Este parâmetro indica que o tipo de *pull* deve ser anónimo.

- `start_soap_server=yes`

  Este parâmetro indica que o servidor de SOAP deve arrancar em conjunto com a LEG.

- `server_url=http://policyserver.nosmadeira.pt/sm/index/soap`

  Para que o servidor de SOAP funcione é necessário especificar o endereço para o *policy server*.

Configuração do /etc/sysctl.conf

- `net.ipv4.ip_local_port_range = 1024 65535`

  Como recomendado pelo manual da SOAP LEG, convém aumentar o limite de portas das portas efémeras.

Configuração rápida

Este *script* não é idempotente – assume uma instalação por defeito e após a primeira aplicação poderá ter efeitos adversos.

```
1  #!/bin/bash
2  P3SM_FILE="/opt/pcube/sm/server/root/config/p3sm.cfg"
3  sed -i "s/introduction_mode=push/introduction_mode=pull/" $P3SM_FILE
4  sed -i "0,/#start=no/{s/#start=no/start=yes/}" $P3SM_FILE
5  sed -i "s/support_ip_ranges=no/support_ip_ranges=yes/" $P3SM_FILE
6  sed -i "s/max_range_size=256/max_range_size=262144/" $P3SM_FILE
7  # Mudar o modelo para o modelo a utilizar
8  echo "[SCE.SCE2000]" >> $P3SM_FILE
9  # Mudar o IP para o SCE a utilizar
10 echo "ip=10.0.0.1" >> $P3SM_FILE
11 echo "port=14374" >> $P3SM_FILE
12
13 SOAP_LEG_FILE="/opt/pcube/sm/server/root/config/soap_leg.cfg"
14 SOAP_URL="http://policyserver.nosmadeira.pt/sm/index/soap"
15 sed -i "s/start=no/start=yes/" $SOAP_LEG_FILE
16 sed -i "s/#supported_pull_type=anonymous/supported_pull_type=anonymous/"
       $SOAP_LEG_FILE
17 sed -i "s/#start_soap_server=no/start_soap_server=yes/" $SOAP_LEG_FILE
```

```
18 sed -i "s|# server_url=|server_url=$SOAP_URL|" $SOAP_LEG_FILE
19
20 echo "net.ipv4.ip_local_port_range" = 1024 65535 >> /etc/sysctl.conf
21 # Mudar limite sem reiniciar
22 echo 1024 65535 > /proc/sys/net/ipv4/ip_local_port_range
```