Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

José Luís Caldeira Pinto

API RESTful para Sistema de Gestão de Desempenho de Redes de Telecomunicações

Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

José Luís Caldeira Pinto

API RESTful para Sistema de Gestão de Desempenho de Redes de Telecomunicações

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Hélder Troca Zagalo, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Dr. Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Dr. Hélder Troca Zagalo
Professor Auxiliar da Universidade de Aveiro

Prof. Dr. Fernando Joaquim Lopes Moreira
Professor Associado da Universidade Portucalense

agradecimentos /
acknowledgements

Em primeiro lugar gostaria de agradecer ao meu orientador, o Professor Doutor Hélder Troca Zagalo, e ao meu supervisor na Nokia Networks, o Engenheiro Nuno Miguel Tavares de Sousa, por todo o apoio prestado.
Gostaria de também agradecer à Nokia Networks por todas as condições que me ofereceu para desenvolver este trabalho e às pessoas que fazem parte da equipa do Performance Manager por toda a ajuda prestada e por todo o conhecimento que me transmitiram.
Por fim, não podia deixar de agradecer à minha família e aos meus amigos por todo o incentivo dado.

Palavras Chave API, REST, RESTful, Performance Manager

Resumo

Durante muito tempo, as organizações restringiram os seus dados e serviços a certas unidades dentro das próprias organizações. A procura por novas formas de negócio e otimização de recursos levaram as organizações a integrarem os seus sistemas e a disponibilizar os dados e serviços a terceiros, através de APIs. As APIs têm vindo a ter uma importância acrescida ao longo dos anos, ao ponto de terem assumido um papel fulcral no sucesso de diversas organizações de renome mundial. A esta tendência não é alheio o surgimento das APIs RESTful, que, derivado das suas vantagens, vieram responder a algumas das necessidades veementes das organizações. Este trabalho surge numa colaboração entre a Universidade de Aveiro e a Nokia Networks e visa o desenvolvimento de uma API RESTful para o Performance Manager. O Performance Manager é uma aplicação de gestão de desempenho de uma rede de telecomunicações, desenvolvida pela Nokia.

Keywords API, REST, RESTful, Performance Manager

Abstract

For a long time, organizations have confined their data and services only to some departments inside the organizations themselves. The quest for new business opportunities and resources optimization led organizations to integrate their systems and make the data and services available to third parties, through APIs. The importance of APIs has been rising to the point that they played a key role in the success of some renowned organizations. The emergence of RESTful APIs contributed to a higher use of APIs inside the organizations, and due to their advantages, they brought answers to some important needs in the organizations. This work arises from a collaboration between the University of Aveiro and Nokia Networks, and it aims for the development of a RESTful API for Performance Manager. Performance Manager is an application, developed by Nokia, for the management of the performance of a telecommunications network.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

ABNF	Augmented Backus-Naur Form
ALPS	Application-Level Profile Semantics
API	Application Programming Interface
ASN.1	Abstract Syntax Notation number One
BML	Business Management Layer
BTS	Base Transceiver Station
CI	Continuous Integration
CORBA	Common Object Request Broker Architecture
CSP	Communication Service Provider
DSL	Domain Specific Language
EML	Element Management Layer
FCAPS	Fault, Configuration, Accounting, Performance, Security
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HAL	Hypertext Application Language
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hyper Text Mark-up Language
HTTP	Hyper Text Transfer Protocol
ITU-T	ITU Telecommunication Standardization Sector
JAX-RS	Java API for RESTful Services
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
JSON-LD	JSON for Linking Data
KPI	Key Performance Indicator
LLA	Logical Layered Architecture
LTE	Long Term Evolution
NML	Network Management Layer
NPM	Nokia Performance Manager

OSS	Operations Support System
PM	Performance Manager
POJO	Plain Old Java Object
PSTN	Public Switched Telephone Network
RAML	RESTful API Modeling Language
REST	Representational State Transfer
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SIREN	Structured Interface for Representing Entities
SOAP	Simple Object Access Protocol
SML	Service Management Layer
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
TDD	Test Driven Development
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TMN	Telecommunications Management Network
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
WADL	Web Application Description Language
WSDL	Web Services Description Language
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

CAPÍTULO 1

INTRODUÇÃO

1.1 CONTEXTO E MOTIVAÇÃO

Um dos desenvolvimentos mais importantes na história humana foi, sem sombra de dúvidas, a revolução industrial. A revolução industrial foi um período de desenvolvimentos, quase simultâneos, em áreas como engenharia mecânica, química, metalurgia e outras disciplinas. De entre todos os desenvolvimentos, a tecnologia mais importante e que motivou a revolução industrial foi o motor a vapor. O motor a vapor permitiu-nos ultrapassar os limites físicos da força humana e animal e gerar enormes quantidades de energia, desencadeando o estilo de vida moderno. A revolução industrial foi a primeira vez, na história, que o progresso da humanidade se deveu primariamente a inovações tecnológicas. Atualmente, estamos a viver uma era em que os avanços tecnológicos estão a ter os mesmos efeitos, no plano mental, que o motor a vapor teve, no plano físico. Devido à importância que a capacidade mental tem no progresso e desenvolvimento, sendo tão ou mais importante que a capacidade física, os avanços tecnológicos digitais deverão contribuir para impulsionar a humanidade, tal como a revolução industrial contribuiu.

Por trás de alguns dos avanços tecnológicos mais significativos, que ocorreram nos últimos anos e que estão a ocorrer neste momento, encontram-se as APIs. A Internet das coisas, as aplicações móveis, as redes sociais ou a computação em nuvem são exemplos de tecnologias que recorrem amplamente a APIs *Web*. A relevância que as APIs estão a ter, atualmente, no mundo digital, também se pode verificar pelo crescimento dramático das APIs *Web* públicas. Na figura 1.1 pode-se ver o crescimento que as APIs *Web* públicas tiveram desde 2005 até 2013. Apesar da evolução fantástica, estes números representam apenas a ponta do icebergue, no que concerne às APIs existentes, porque não estão contabilizadas as APIs privadas.

O ponto de partida para o uso de APIs no mundo empresarial deu-se quando as organizações começaram a sentir a necessidade de integrarem os seus sistemas internos. Após abordarem esta necessidade, as organizações começaram a tentar encontrar novas formas de explorar o conhecimento que possuíam tendo em vista novos modelos de negócio. Esta busca levou as organizações a disponibilizarem

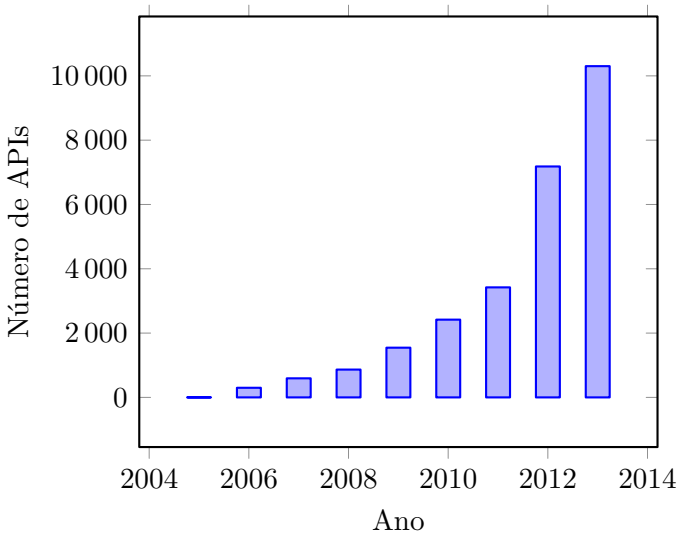

Figura 1.1: Crescimento do número de APIs *Web* ao longo dos anos (Fonte: ProgrammableWeb.com)

os dados e serviços, que possuíam exclusivamente para consumo interno, a terceiros. Este caminho trilhado pelas organizações contribuiu fortemente para o crescimento do número de APIs que se verificou nos últimos anos. As APIs alteraram de tal modo a realidade das organizações que hoje em dia, as APIs, são olhadas como uma vantagem competitiva e não como um pormenor técnico que apenas diz respeito aos programadores.

Naturalmente, o tipo ou estilo das APIs desenvolvidas foi evoluindo ao longo dos anos. O conceito por detrás das APIs já existe desde os primórdios da computação mas só a partir do ano 2000 é que começaram a surgir as primeiras APIs *Web*. Nos últimos anos, o estilo predominante para o desenvolvimento de APIs *Web* tem sido o Representational State Transfer (REST), como se pode ver na figura 1.2.

Para além de disponibilizarem serviços e dados, para o público, através de APIs públicas, que implementam o estilo REST, as organizações estão, cada vez mais, a usar este estilo de APIs para integrarem os seus sistemas. É neste sentido que a Nokia Networks também está a caminhar, sendo parte integrante de uma iniciativa para modernizar algumas das suas soluções. Esta dissertação surge precisamente no contexto da modernização de uma das aplicações da Nokia Networks, o Performance Manager (PM).

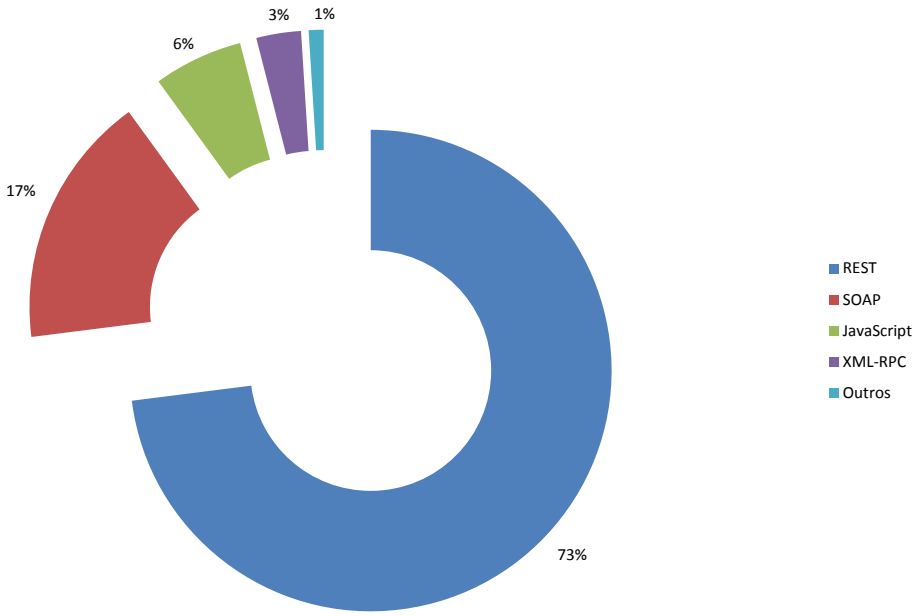

Figura 1.2: Distribuição dos estilos de APIs (Fonte: ProgrammableWeb.com, Maio de 2011)

1.2 OBJETIVOS

O objetivo principal deste trabalho é o desenvolvimento de uma Application Programming Interface (API), que segue o estilo REST, para um sistema de gestão de desempenho de redes de telecomunicações, o PM. Pretende-se que esta API, em primeira instância, suporte a funcionalidade mais importante para os clientes do PM, que é possibilidade de obter valores de indicadores de desempenho de uma rede de telecomunicações. Para isso, a API deve disponibilizar todos os recursos necessários para que um cliente possa obter esses valores. Espera-se que esta API venha a substituir a API existente atualmente e que seja usada quer pelo novo cliente Hyper Text Mark-up Language (HTML) 5, como por clientes móveis ou por outras aplicações que tenham que recorrer ao PM. Também é objetivo que a API seja desenvolvida de acordo com os padrões de qualidade e metodologias da Nokia usando várias técnicas para tal, nomeadamente TDD.

1.3 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação encontra-se dividida em seis capítulos.

Capítulo 1: Introdução No capítulo 1, o presente capítulo, é apresentado o contexto, a motivação, os objetivos e a estrutura da presente dissertação.

Capítulo 2: Estado da Arte No capítulo 2 é apresentado o conhecimento atual na área dos sistemas de suporte à operação de uma rede de telecomunicações e é descrito o sistema de suporte à operação disponibilizado pela Nokia, com ênfase na solução de gestão de desempenho. Também é feito uma introdução ao Hyper Text Transfer Protocol (HTTP), devido à sua relevância para

a implementação de uma API RESTful, e é apresentado o conhecimento atual na área dos serviços *Web*, na qual se insere o REST.

Capítulo 3: Requisitos da API No capítulo 3 é contextualizada a API e descrito o domínio do problema. Adicionalmente são apresentados os requisitos da API, desde os casos de uso até às restrições impostas à API.

Capítulo 4: Desenho da API No capítulo 4 são apresentados os aspetos que tiveram que ser tidos em conta para desenhar a API e é feita uma discussão das possíveis soluções para cada aspeto e a razão subjacente à solução escolhida.

Capítulo 5: Implementação da API No capítulo 5 é descrito, ao nível das classes de *software*, uma pequena parte da API implementada e são apresentadas as metodologias e tecnologias usadas para implementar a API.

Capítulo 6: Resultados e Conclusão No capítulo 6 é discutido o que foi alcançado com este trabalho, os obstáculos encontrados e possível trabalho futuro.

CAPÍTULO 2

Estado da Arte

Este capítulo começa por apresentar os sistemas onde a API se insere e posteriormente são introduzidos os temas relacionados com APIs *Web*.

2.1 sistemas de suporte à operação

Hoje em dia, os clientes de uma operadora de telecomunicações esperam que os serviços (voz, multimédia, mensagens, etc.), disponibilizados por esta, tenham um nível elevado de qualidade e desempenho e estejam disponíveis em qualquer momento, lugar e dispositivo. Na área das telecomunicações, a percentagem de tempo que os sistemas e serviços estão disponíveis assume grande importância, ao ponto das operadoras de telecomunicações almejarem uma disponibilidade de 99.999%, ou cinco noves, como é usualmente referido [1]–[3]. Consequentemente, os operadores necessitam de suportar diferentes tecnologias para poder oferecer os serviços, com as condições referidas anteriormente e com o menor custo de operação possível, para um conjunto de clientes que pode ser numeroso e estar bastante disperso geograficamente. Paralelamente, é comum um operador de telecomunicações recorrer a mais que um fabricante de equipamentos de telecomunicações, levando a que a sua rede de telecomunicações seja composta por equipamentos de diferentes fabricantes. Resulta assim, a necessidade de existir uma interoperabilidade, a diversos níveis, entre os equipamentos de telecomunicações de diferentes fabricantes. Todos estes fatores podem levar a que a rede de telecomunicações de uma operadora seja bastante complexa, podendo ser composta por muitos tipos de equipamentos de telecomunicações e equipamentos de suporte, que convencionalmente são designados de elementos de rede. Um exemplo desta complexidade pode ser visualizado na figura 2.1 que mostra a arquitetura de uma rede de telecomunicações móveis que suporta diferentes tecnologias.

No exemplo da figura 2.1, a rede está organizada em dois sectores: a rede nuclear e a rede de acesso celular. Sucintamente, pode-se dizer que a rede de acesso celular é responsável por interligar os equipamentos dos clientes e a rede nuclear. A rede nuclear interliga as diferentes redes de acesso e oferece diversos serviços aos clientes. Para além disso, a rede nuclear também comunica com outras redes externas, como é o caso da Public Switched Telephone Network (PSTN). Como foi dito anteriormente, este exemplo mostra uma rede de telecomunicações que suporta as tecnologias Global System for Mobile Communications (GSM) (2G), General Packet Radio Service (GPRS)

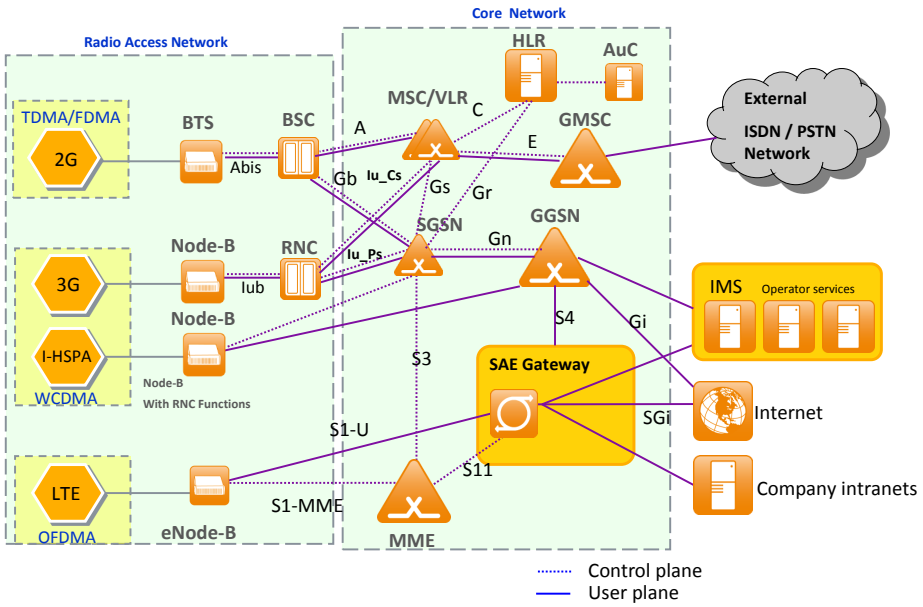

Figura 2.1: Arquitetura de uma rede de telecomunicações móveis

(2.5G), Universal Mobile Telecommunications System (UMTS) (3G) e Long Term Evolution (LTE). A arquitetura de cada uma destas tecnologias é composta por diversos elementos que comunicam entre si. Cada um destes elementos desempenha funcionalidades específicas, podendo dar-se o exemplo do elemento Base Transceiver Station (BTS), nas tecnologias GSM e GPRS, que é responsável pela transmissão e receção do sinal de rádio. Na rede, para além do tráfego gerado pelos clientes, também circulam mensagens de sinalização para controlar, entre outras coisas, as comunicações dos clientes. Na figura pode-se ver os canais responsáveis por transportar os dados dos clientes (linha sólida) e os de sinalização (linha tracejada).

Para fazer face aos requisitos anteriormente expostos, os operadores de telecomunicações recorrem a sistemas de suporte à operação (OSSs). Um sistema de suporte à operação (OSS - Operations Support System) é constituído por um conjunto de aplicações integradas que suportam o desenho, construção e funcionamento de uma rede de telecomunicações e dos serviços que assentam sobre a rede. Para isso, os OSSs têm que ser integrados nas redes de telecomunicações de modo a comunicar com os elementos de rede. Os objetivos de um OSS passam por garantir que a rede é eficiente, os serviços são rentáveis e os clientes estão satisfeitos.

Devido à inerente complexidade de uma rede de telecomunicações, um OSS apresenta diversos desafios na sua implementação. Para fazer face a estes desafios, o ITU Telecommunication Standardization Sector (ITU-T) definiu o conceito de Telecommunications Management Network (TMN) e desenvolveu um conjunto de recomendações relacionadas com o mesmo (TMN series). Uma TMN, nos termos do ITU-T, fornece funcionalidades de gestão para as redes e serviços de telecomunicações e oferece comunicações entre si e as redes de telecomunicações, serviços e outros TMNs [4]. De seguida são apresentados alguns conceitos presentes no conjunto de recomendações TMN.

2.1.1 LLA - LOGICAL LAYERED ARCHITECTURE

Um dos conceitos mais importantes presente nas normas TMN foca-se na organização das funcionalidades de gestão de uma rede de telecomunicações. De modo a lidar com a complexidade de uma rede de telecomunicações, o ITU-T introduziu um modelo de nome Logical Layered Architecture (LLA) que agrupa as funcionalidades de gestão em camadas lógicas e descreve a relação entre as camadas. Este modelo de referência tem na sua composição uma camada de gestão de negócio, uma camada de gestão de serviços, uma camada de gestão de rede e uma camada de gestão de elementos de rede. A divisão das funcionalidades nas camadas referidas anteriormente, apesar de ter grande aceitação, não deve ser vista como a única solução, podendo existir outras soluções com um agrupamento de camadas diferentes [4].

CAMADA DE GESTÃO DE NEGÓCIO

Todas as camadas do modelo LLA, à exceção da camada de gestão de negócio (BML - Business Management Layer), focam-se em otimizar a utilização dos recursos de telecomunicações existentes. Contrariamente às outras camadas, o foco da camada de gestão de negócio centra-se na otimização do investimento e utilização de novos recursos. Além de suportar o processo de decisão de investimento, esta camada também suporta outros aspetos de gestão dos recursos de uma empresa, mantendo para isso dados agregados de toda a empresa.

CAMADA DE GESTÃO DE SERVIÇOS

A camada de gestão de serviços (SML - Service Management Layer) é responsável pelos aspetos contratuais dos serviços que são fornecidos aos clientes. As principais funções desta camada são a criação de serviços, implementação de serviços, monitorização de serviços, processamento de pedidos e queixas dos clientes e faturação.

CAMADA DE GESTÃO DE REDE

Com a ajuda das funcionalidades da camada de gestão de elementos de rede (EML - Element Management Layer), a camada de gestão de rede (NML - Network Management Layer) oferece uma visão holística da rede que pode estar dispersa por uma grande área geográfica e pode ser constituída por variadas tecnologias. Através do conhecimento e controlo dos recursos da rede, esta camada tem a capacidade de gerir o desempenho da rede.

CAMADA DE GESTÃO DE ELEMENTOS DE REDE

A camada de gestão de elementos de rede (EML) providencia funcionalidades para a gestão dos elementos individuais da rede ou grupos de elementos da rede.

2.1.2 FCAPS

Um fornecedor de serviços de comunicações (CSP - Communication Service Provider), para garantir que a sua rede de telecomunicações é operada eficientemente, necessita de abordar um vasto conjunto de processos de gestão de rede. Tendo isto em vista, o ITU-T estendeu as normas TMN para incorporar o modelo Fault, Configuration, Accounting, Performance, Security (FCAPS). O modelo FCAPS agrupa a gestão de uma rede de telecomunicações em cinco áreas funcionais denominadas por falha, configuração, contabilidade, desempenho e segurança (FCAPS - Fault, Configuration, Accounting, Performance, Security). Cada camada do modelo LLA oferece, de alguma forma, todas as funcionalidades do modelo FCAPS ou um subconjunto dessas funcionalidades, como pode ser visto na figura 2.2. Importa salientar, com recurso à figura, que cada camada do modelo LLA depende dos serviços oferecidos pela camada adjacente inferior [5].

FALHA

A gestão de falhas oferece um conjunto de funcionalidades que permitem a deteção, isolamento e correção de anomalias na rede de telecomunicações.

CONFIGURAÇÃO

A gestão da configuração providencia as funcionalidades necessárias para controlar e configurar os elementos da rede de telecomunicações.

CONTABILIDADE

A gestão da contabilidade permite medir a utilização dos serviços fornecidos por uma rede de telecomunicações e determinar os custos para os fornecedores dos serviços e para os clientes pela utilização desses serviços.

DESEMPENHO

Através da obtenção de dados estatísticos da rede e subsequente análise, a gestão de desempenho permite avaliar e reportar o comportamento dos equipamentos de telecomunicações e a eficiência da rede e também pode auxiliar o planeamento de uma rede. A gestão de desempenho é um processo tão crítico como a gestão de falhas porque a degradação de um serviço pode ter um impacto para o cliente tão grande como uma falha na rede.

SEGURANÇA

Numa rede é necessário garantir que os clientes só têm acesso a recursos para os quais estão devidamente autorizados e também é necessário assegurar que apenas pessoas devidamente autorizadas podem configurar os equipamentos e serviços da rede.

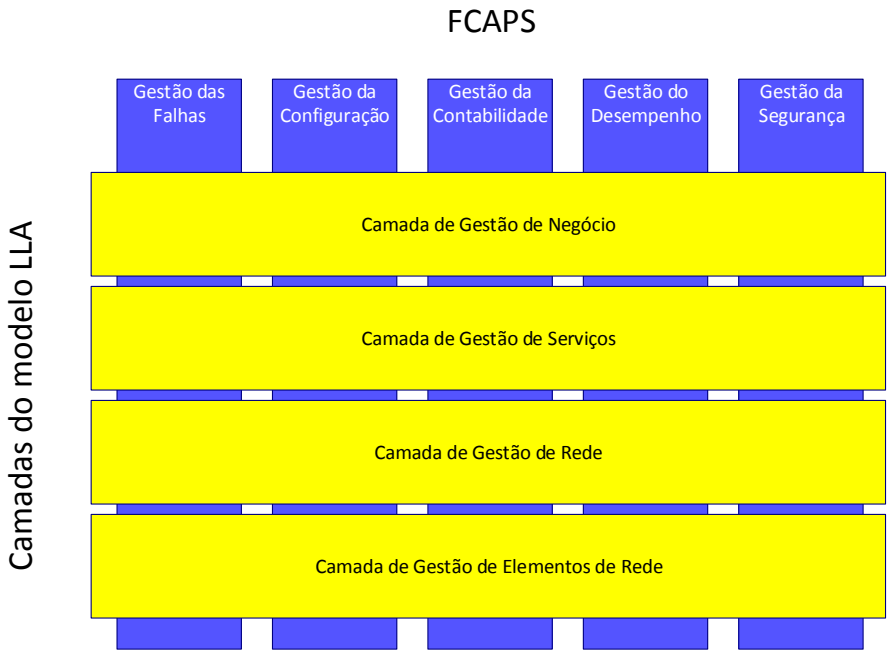

Figura 2.2: Relação entre os modelos LLA e FCAPS

2.1.3 NETACT

Com mais de 400 instalações, o NetAct é o OSS de referência da Nokia. Parte do sucesso do NetAct deve-se à preocupação em ir ao encontro das prioridades manifestadas pelos CSPs, nomeadamente em oferecer serviços que melhorem a experiência do cliente, sendo que, em mercados desenvolvidos, a qualidade é o aspeto mais relevante para essa experiência. Baseado em normas, algumas das quais referidas anteriormente, o NetAct providencia, sobre uma plataforma, um conjunto de aplicações que satisfazem as necessidades dos CSPs.

O NetAct pode ser visto como um sistema constituído por três camadas em que a camada base é uma plataforma que fornece os recursos, de *hardware* e *software*, que suportam as camadas superiores. A camada de adaptação e mediação fornece a interface através da qual os dados originados nos elementos da rede entram no NetAct e podem ser manipulados pelas aplicações do NetAct. Para se compreender melhor esta camada, importa referir em que consiste uma adaptação e uma mediação. Uma adaptação consiste em meta-dados que são usados para configurar as aplicações e mediações de modo a suportarem novos elementos de rede. Uma mediação implementa as conversões necessárias dos dados entre os elementos de rede e o NetAct. Estas conversões podem acontecer entre os protocolos de rede, como por exemplo Simple Network Management Protocol (SNMP), ou formatos dos dados, como por exemplo Abstract Syntax Notation number One (ASN.1), e os processos e formatos internos do NetAct. Por último, a camada de aplicação contém as aplicações necessárias para realizar as tarefas comuns de gestão, como por exemplo gestão de falhas, gestão de configuração, gestão de performance e otimização da rede. A figura 2.3 mostra como é que as camadas referidas anteriormente se organizam.

Devido à dimensão de uma rede, tipo de rede, topologia de uma rede ou necessidades organizacionais de um operador, uma rede pode ter que ser dividida em várias sub-redes. Para cada uma destas sub-redes existe um *cluster* regional que gere todos os detalhes da sub-rede. É neste *cluster* regional que está inserido o NetAct. Acima destes *clusters* regionais pode existir um *cluster* global que oferece

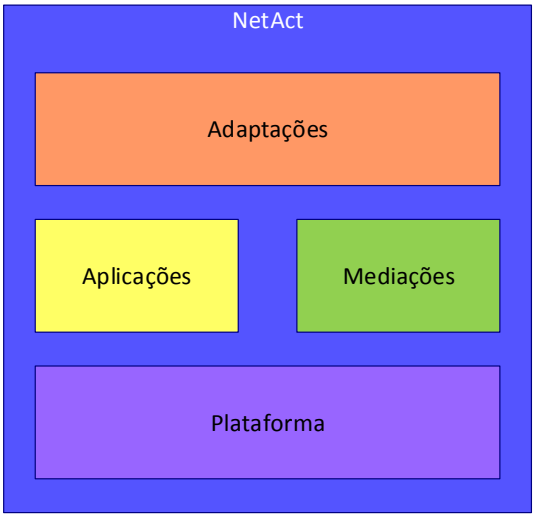

Figura 2.3: Arquitetura do NetAct

uma visão alargada de toda a rede. Neste *cluster* global também existe o NetAct, mas numa versão que contém bastante menos aplicações que o NetAct do *cluster* regional. Na figura 2.4 pode-se ver esta organização em *clusters*.

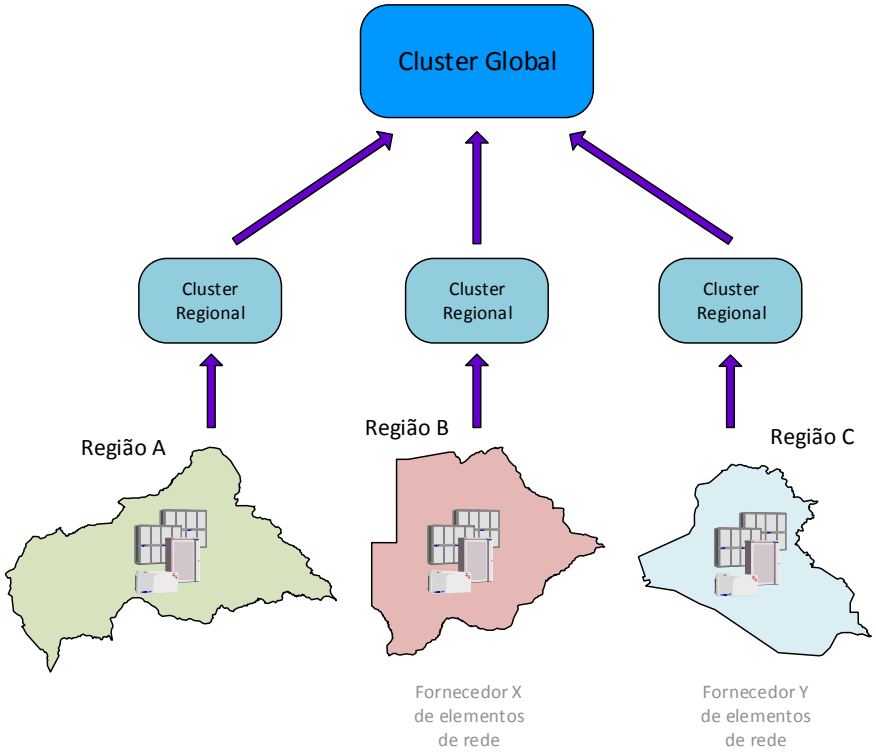

Figura 2.4: Organização do NetAct em clusters

2.2 GESTÃO DE DESEMPENHO

Os operadores de telecomunicações para fornecerem serviços, com valor para os clientes, necessitam de ter uma visão do desempenho da rede que suporta esses serviços. Esta informação pode estar dispersa, ser de difícil acesso e não apresentar uma visão consistente e holística da rede. Isto pode levar a que o operador de telecomunicações não opere com a eficiência desejada, culminando em serviços de baixa qualidade para os clientes. Um sistema de gestão de desempenho que colete os dados da rede, os interprete e os apresente de uma forma unificada para os operadores da rede, permite a deteção de problemas na rede antes destes ocorrerem. Deste modo, a gestão de desempenho assume um papel importante nas operações diárias de uma rede de telecomunicações. Do ponto de vista financeiro, as vantagens de um sistema de gestão de desempenho podem traduzir-se em reduções nas despesas operacionais (OPEX) e na otimização das despesas de capital (CAPEX). As reduções nas despesas operacionais podem ser alcançadas através da otimização dos recursos, automação de processos e maior eficiência operacional. Como exemplo, relatórios rigorosos de desempenho resultam, usualmente, numa redução, entre 10% a 50%, do tempo necessário para planear a otimização de uma rede. A otimização das despesas de capital pode ser alcançada através de uma melhor previsão das necessidades de evolução e expansão de uma rede de telecomunicações.

Gerir o desempenho de uma rede de telecomunicações é uma tarefa complexa. Esta complexidade está patente na tabela 2.1, que mostra as necessidades, de cada tipo de utilizador, que devem ser satisfeitas por um sistema de gestão de desempenho.

Grupo de utilizadores	Objetivos da gestão do desempenho
Planeamento e otimização	Monitorizar a capacidade da rede Monitorizar a qualidade da rede Verificar a cobertura da rede
Operações e manutenção	Detetar e resolver falhas Descobrir áreas potencialmente problemáticas Monitorizar estatísticas da rede Monitorizar expansões da rede e atualizações dos sistemas
Gestão	Obter relatórios sumariados da qualidade da rede Prever tendências futuras
Apoio ao cliente	Informar os clientes de eventuais problemas Responder de forma mais célere a queixas relacionadas com os serviços
Marketing	Obter informações relevantes sobre os comportamentos dos clientes Planear novos serviços

Tabela 2.1: Funcionalidades de um sistema de gestão de desempenho

2.2.1 PERFORMANCE MANAGER

Tendo em vista as necessidades dos operadores de telecomunicações, referidas anteriormente, e aproveitando o conhecimento adquirido ao longo dos anos no mercado das telecomunicações, a Nokia tomou a decisão de desenvolver uma solução de gestão de desempenho. Esta solução de gestão de desempenho foi evoluindo ao longo dos anos até chegar à solução existente atualmente, o PM.

O PM é uma solução centralizada de gestão de desempenho da rede que coleta e guarda dados de desempenho da rede e oferece uma visão holística e agnóstica do desempenho da rede. Além de dados de desempenho, também são guardados outros dados relevantes, tais como dados de configuração da rede, estatísticas de falhas na rede e dados relacionados com os serviços fornecidos. Esta coleção de dados de diversos domínios permite analisar, por exemplo, se a capacidade de uma rede configurada num dado momento é suficiente face às exigências. Uma das funcionalidades chave do PM é a agregação dos dados de diversas formas de modo a permitir uma análise eficaz, evitando-se guardar os dados em bruto por longos períodos de tempo. Para esse efeito, existe um conjunto variado de agregações possíveis de serem efetuadas, quer ao nível do tempo, quer ao nível dos elementos da rede. Estes dados agregados podem ser guardados por longos períodos e serem usados no cálculo de indicadores chave de desempenho (KPI - Key Performance Indicator) e na produção de relatórios. O PM é fornecido aos operadores com um vasto conjunto pré-definido de relatórios e KPIs que satisfazem quase todas as necessidades dos operadores.

Atualmente o PM está disponível em duas versões sendo que a versão mais básica é oferecida conjuntamente com o NetAct, sendo denominada por NetAct PM. Apesar de esta versão satisfazer grande parte das necessidades dos operadores, existem certos casos que necessitam de uma solução mais avançada de gestão de desempenho. Face a isto, a Nokia oferece uma versão mais avançada, de

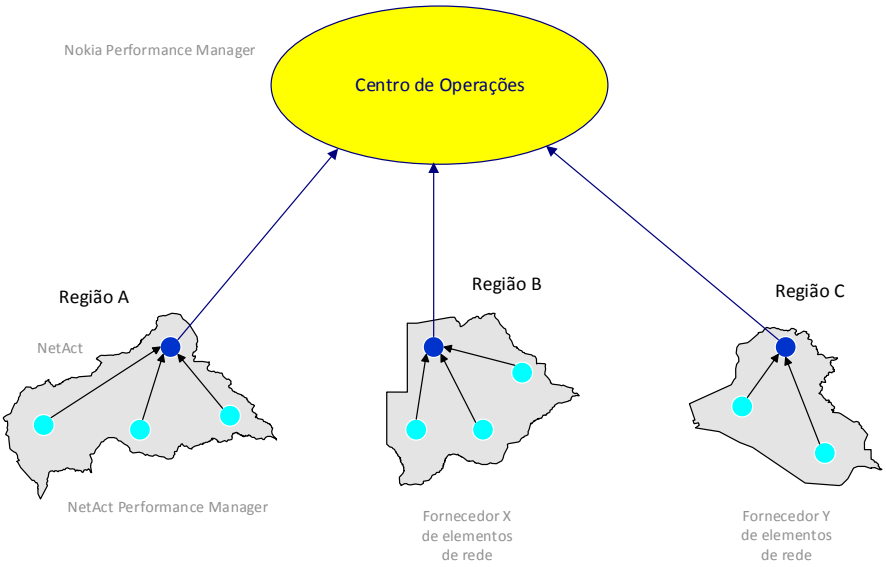

Figura 2.5: Contextualização do NetAct PM e do NPM

seu nome Nokia Performance Manager (NPM). Entre outros aspetos diferenciadores em relação ao NetAct PM, o NPM permite guardar uma maior quantidade de dados coletados da rede e por um período mais alargado. Outro foco do NPM é o suporte de múltiplos fornecedores de elementos de rede, fazendo face ao panorama atual em que as redes dos operadores são compostas por equipamentos de diversos fornecedores (Nokia, Ericsson, Huawei, entre outros).

Esta diferenciação do PM em duas versões está relacionado com o posicionamento do PM ao nível dos *clusters* apresentados na figura 2.4. Em cada *cluster* regional existe o NetAct PM, enquanto que o NPM está orientado para um *cluster* global. Na figura 2.5 pode-se visualizar um exemplo de um operador de telecomunicações que possui uma rede de telecomunicações móveis dividida em três regiões, sendo que em cada região existe um *cluster* regional onde está instalado o NetAct e o NetAct PM. Também se pode dar o caso de uma região ter um fornecedor de elementos de rede diferente das outras regiões. Isto leva a que o operador, para ter uma visão global de toda a rede, necessite de ter um centro de operações de rede. Com o objetivo a gerir o desempenho de toda a rede, no centro de operações pode existir um *cluster* global com o NPM instalado.

Atualmente, o NetAct PM expõe serviços para outras aplicações da Nokia através de uma API. Esta API apresenta problemas ao nível do desempenho e qualidade do código e conjuntamente a estes problemas, surgiram novos requisitos funcionais que necessitam de ser implementados. Paralelamente, existe a necessidade de evoluir a interface gráfica do NPM de modo a dar resposta aos clientes da Nokia. As tecnologias usadas na interface gráfica (Flex) e na comunicação desta com o servidor (BlazeDS) deixaram de ser desenvolvidas pela Adobe e já não suportam os últimos dispositivos móveis. Além disto, também é necessário oferecer novas funcionalidades que não são suportadas pelas tecnologias atualmente em uso. Todos estes fatores levaram à decisão de migrar a aplicação para HTML 5. Sendo o BlazeDS uma tecnologia direcionada para as aplicações desenvolvidas em Flex e AIR, existe a necessidade de criar uma nova API que forneça os serviços necessários ao cliente HTML 5.

2.3 HTTP

O HTTP é um protocolo, que se enquadra na camada aplicacional do modelo TCP/IP, para as comunicações entre sistemas distribuídos. Caracteriza-se por ser um protocolo genérico, podendo ser usado em diversos contextos, que providencia uma interface uniforme para interagir com os recursos através da manipulação e transferência de representações. Esta interface é independente dos recursos fornecidos, ou seja, não depende da natureza do recurso ou da sua implementação, permitindo assim esconder os detalhes de implementação de um serviço. [6]–[9]

O protocolo HTTP foi evoluindo ao longo dos anos, tendo na 0.9 a sua primeira versão, até chegar à versão atual, a 2.0. Seguidamente, é explicada alguma da semântica do protocolo HTTP. De notar que esta semântica baseia-se na versão 1.1 do protocolo, que se mantém na versão 2.0.

2.3.1 MENSAGENS

As comunicações no protocolo HTTP processam-se através de mensagens. Uma mensagem que é enviada de um cliente para um servidor é classificada como um pedido HTTP e uma mensagem que é enviada de um servidor para um cliente é classificada como uma resposta HTTP. No contexto do HTTP, um cliente é um programa que estabelece uma conexão a um servidor com o propósito de enviar um ou mais pedidos HTTP e um servidor é um programa que aceita conexões de modo a receber os pedidos HTTP e enviar as respostas HTTP. Os termos cliente e servidor referem-se apenas aos papéis que estes programas desempenham para uma conexão em particular, podendo acontecer que o mesmo programa aja como cliente em certas conexões e noutras aja como servidor. O HTTP é usualmente referido como sendo um protocolo sem estado. O que isto significa é que cada transação é atómica, não havendo nada na especificação que exija algum tipo de associação entre diferentes transações. Uma transação consiste num único pedido HTTP e na resposta HTTP correspondente.

Um pedido HTTP é composto por três elementos. O primeiro elemento é uma linha que especifica o método, Uniform Resource Identifier (URI) e versão do protocolo HTTP usada. O elemento seguinte, o cabeçalho, consiste em campos que incluem informação de suporte, que pode ajudar a perceber melhor o pedido. Por fim, o último elemento é o corpo da mensagem. É no corpo de uma mensagem que se encontra normalmente a representação de um recurso. A maior parte dos pedidos HTTP não contêm o corpo, porque, usualmente, o propósito de um pedido HTTP é requisitar conteúdo. Uma resposta HTTP, à semelhança de um pedido, também é composta por três elementos. O primeiro elemento é uma linha que especifica a versão do protocolo HTTP usada, um código de estado e uma breve descrição do código de estado. O elemento seguinte, o cabeçalho, consiste em campos que fornecem informação relacionada com a natureza da resposta. Por fim, o último elemento é o corpo da mensagem.

2.3.2 RECURSOS

Do ponto de vista do consumidor de um recurso, um recurso é algo com que o consumidor interage enquanto progride para alcançar um determinado objetivo. O HTTP não restringe a natureza dos recursos. Quase tudo pode ser modelado como um recurso e disponibilizado na rede para ser manipulado. Os recursos podem ser páginas *Web*, imagens, catálogos de produtos, entre outros exemplos. Para

usar um recurso é necessário uma maneira de o identificar na rede e manipular. Para este propósito, o HTTP usa o identificador uniforme de recurso (URI - Uniform Resource Identifier). Um URI identifica unicamente um recurso e ao mesmo tempo torna-o endereçável, ou seja, torna possível a sua manipulação.

2.3.3 REPRESENTAÇÕES

Uma representação é uma transformação ou vista do estado de um recurso num dado instante de tempo. Esta vista é codificada em um ou mais formatos passiveis de serem transferidos, tais como Extensible Markup Language (XML) ou JavaScript Object Notation (JSON). O acesso a um recurso é feito sempre através das suas representações, isto é, os recursos nunca são acedidos diretamente. Do ponto de vista do HTTP, uma representação é informação que reflete o estado de um recurso, num formato que pode ser comunicado através do protocolo HTTP e que consiste num conjunto de meta-dados da representação e nos dados efetivos da representação. Um servidor pode ser capaz de gerar ou aceitar múltiplas representações para o mesmo recurso, o que leva a que o servidor tenha que selecionar, de alguma forma, a representação mais apropriada. A forma de selecionar esta representação, normalmente, é baseada numa negociação de conteúdo entre o servidor e cliente.

2.3.4 TIPOS DE MÉDIA

Existe um conjunto de campos no cabeçalho de uma mensagem HTTP que ajudam o cliente e o servidor a interpretar o conteúdo que é transportado no corpo da mensagem. No campo *Content-Type*, tal como no campo *Accept*, é indicado qual o tipo de média da representação associada, usando para isso, uma especificação de tipos de média que foi desenhada para ser extensível [10]. O tipo de média define o formato dos dados e como é que os dados devem ser manipulados pelo recipiente da mensagem HTTP.

2.3.5 MÉTODOS

Os métodos indicam o propósito dos pedidos feitos pelos clientes. Cada pedido HTTP tem um método, método esse que diz ao servidor qual a ação a executar. A especificação do HTTP define os métodos apresentados na tabela 2.2.

Na tabela 2.2, além dos métodos que estão incluídos na especificação do HTTP/1.1, também é apresentado o método PATCH, que foi especificado à posteriori, mas que pode assumir alguma importância no desenho e implementação de uma API RESTful. Este método vem dar resposta à necessidade de modificar parcialmente um recurso, dado que o método PUT é mais apropriado para a substituição completa de um recurso. A sua utilização para modificações parciais pode confundir os *proxies*, *caches* e até os clientes e servidores. O método POST também sofre do mesmo problema do método PUT, não possuindo a semântica apropriada para a substituição parcial de um recurso.

Os métodos HTTP referidos na tabela 2.2 partilham algumas propriedades, de entre as quais, a segurança e a idempotência são as mais relevantes quando se desenha uma API RESTful. A tabela 2.3 resume as propriedades de cada método. Quando um cliente faz um pedido e espera que esse pedido

Método	**Descrição**
GET	Transfere a representação atual do recurso alvo.
HEAD	O mesmo que o GET, mas só transfere o cabeçalho.
POST	Executa algum tipo de processamento, especifico a um recurso, no conteúdo do corpo do pedido.
PUT	Substitui todas as representações atuais do recurso alvo pelo conteúdo do corpo do pedido.
PATCH	Aplica as modificações, parciais, descritas no corpo do pedido, ao recurso alvo.
DELETE	Remove todas as representações atuais do recurso alvo.
CONNECT	Estabelece um túnel para o servidor identificado pelo recurso alvo.
OPTIONS	Descreve as capacidades suportadas pelo recurso alvo.
TRACE	Executa um teste com o intuito de verificar como é que a mensagem é manipulada ao longo do caminho até ao recurso alvo.

Tabela 2.2: Métodos HTTP

não provoque alterações no estado do servidor, diz-se que o método usado pelo cliente é seguro. Um método seguro não previne que o servidor se comporte de uma maneira inesperada, apenas determina que o cliente não pode ser responsabilizado por qualquer problema que ocorra no servidor por este executar código potencialmente inseguro (por exemplo, apagar um registo da base de dados) em resposta a um método seguro. De entre os métodos HTTP já apresentados, apenas o GET, HEAD, OPTIONS e TRACE são seguros. Se ao fazer um pedido com um determinado método, o efeito no servidor for o mesmo que ao fazer múltiplos pedidos idênticos com o mesmo método, então o método é considerado idempotente. Todos os métodos seguros também são idempotentes e, para além desses, o PUT e o DELETE também são idempotentes. Quando um cliente faz um pedido PUT, o estado do servidor no final do pedido é igual ao estado caso o cliente faça vários pedidos PUT sempre com os mesmos dados, pois o método PUT substitui todas as representações atuais do recurso alvo pelas representações enviadas pelo cliente.

Método	Seguro	Idempotente
GET	√	√
HEAD	√	√
POST	×	×
PUT	×	√
PATCH	×	×
DELETE	×	√
CONNECT	×	×
OPTIONS	√	√
TRACE	√	√

Tabela 2.3: Propriedades dos métodos HTTP

2.3.6 CÓDIGOS DE ESTADO DA RESPOSTA

Quando um servidor responde a um pedido de um cliente, inclui na mensagem de resposta um código composto por três dígitos. Este código traduz o resultado da tentativa do servidor em perceber e satisfazer o pedido do cliente. Existe um conjunto alargado de códigos que são categorizados de acordo com a resposta do servidor, tal como mostra a tabela 2.4. Na tabela 2.5 estão referenciados alguns dos códigos de estado mais relevantes no contexto de uma API RESTful.

Classe	Descrição
1xx (Informativo)	O pedido foi recebido, a continuar o processo.
2xx (Bem Sucedido)	O pedido foi recebido com sucesso, compreendido e aceite.
3xx (Redirecionamento)	São necessárias realizar mais ações para completar o pedido.
4xx (Erro do cliente)	O pedido não está bem formulado sintaticamente ou não foi possível atender o pedido.
5xx (Erro do servidor)	O servidor não conseguiu atender um pedido aparentemente válido.

Tabela 2.4: Categorização dos códigos de estado de uma resposta HTTP

Código	Significado
200 Ok	O pedido foi executado com sucesso.
201 Created	O pedido foi executado com sucesso e em resultado foi criado um ou mais recursos que são identificados através do campo Location do cabeçalho da resposta ou através do próprio URI do pedido.
400 Bad Request	Existe um problema no pedido e em virtude disso, o servidor não o vai processar.
403 Forbidden	O pedido foi compreendido mas recusado.
404 Not Found	Não foi encontrada uma representação para o recurso alvo ou o servidor recusa-se a divulgar que existe.
405 Method Not Allowed	O método especificado não é suportado para o recurso indicado pelo pedido.
409 Conflict	Existe um conflito com o estado atual do recurso alvo.
500 Internal Server Error	O servidor encontrou um problema que o impede de prosseguir com o processamento do pedido.

Tabela 2.5: Significado de alguns códigos de estado de uma resposta HTTP

2.3.7 HTTP/2

Apesar do HTTP/1.1 ser um protocolo com bastante sucesso, a forma como o protocolo usa a camada inferior, de transporte, do modelo TCP/IP, tem efeitos negativos no desempenho das aplicações atuais. Em particular, os clientes HTTP/1.1 para conseguirem fazer vários pedidos ao mesmo tempo têm que usar múltiplas conexões. Adicionalmente, os campos do cabeçalho, em grande parte dos casos, acabam por ser repetitivos e verbosos, originando, desnecessariamente, mais tráfego na rede. À medida que as aplicações *Web* foram evoluindo, estas limitações começaram a ser mais notórias tanto para os utilizadores da *Web*, como para os programadores das aplicações *Web* [11], [12].

As limitações de desempenho do HTTP/1.1 levaram a Google a desenvolver e a anunciar em 2009 o protocolo experimental SPDY [13] com o objetivo de reduzir em 50% o tempo de carregamento das páginas *Web*. Após ser testado em laboratório com resultados promissores (redução do tempo de carregamento em 55%), o SPDY começou a ser suportado nos principais browsers e começou a ser usado por diversas companhias nos seus sítios *Web*. Consciente do sucesso crescente do SPDY, o HTTP Working Group começou a trabalhar numa nova norma, denominada HTTP/2, tendo por base o SPDY. Após diversos anos de trabalho, em Maio de 2015 foi publicada a especificação do HTTP/2 [11], [12].

O HTTP/2 foca-se nos aspetos de desempenho do protocolo, mantendo a mesma semântica do HTTP/1.1. Significa isto que os principais conceitos, tais como os métodos, códigos de estado e campos do cabeçalho, não sofreram alterações. Ao invés, o HTTP/2 modifica a maneira como os dados são formatados e transportados entre os clientes e servidores. Os objetivos do HTTP/2 passam por reduzir a latência observada pelo utilizador final, minimizar o *overhead* introduzido pelo protocolo,

adicionar suporte à priorização dos pedidos e suporte ao envio não solicitado de representações dos servidores para os clientes. Para isto, foi introduzido a compressão do cabeçalho e permitido que na mesma conexão possam ocorrer múltiplas transferências de informação em concorrência. O protocolo resultante tira melhor partido da capacidade da rede porque, em comparação com o HTTP/1.1, são necessárias menos conexões TCP e também possibilita um processamento mais eficiente das mensagens [11], [12].

2.4 REST

Tradicionalmente, recorria-se a arquiteturas complexas, tal como o Common Object Request Broker Architecture (CORBA) ou o Remote Method Invocation (RMI), para construir sistemas distribuídos. No entanto, este tipo de soluções apresentavam alguns problemas. Em primeiro lugar, tinha que se recorrer à mesma tecnologia no lado do servidor e do cliente, tecnologia essa que era complexa de implementar. Também não existia a garantia que diferentes implementações da mesma tecnologia fossem interoperáveis. Adicionalmente, estas tecnologias também levantavam problemas de segurança devido à necessidade de abrir portas Transmission Control Protocol (TCP). Finalmente, como o estado era mantido entre diferentes chamadas do cliente, também existiam problemas de escalabilidade. [14]

Para fazer face aos problemas supracitados e face ao sucesso da *Web*, os princípios da arquitetura subjacente à *Web* começaram a ser ponderados para construir outros tipos de sistemas distribuídos. Simple Object Access Protocol (SOAP) foi um dos modelos que ganhou mais notoriedade e durante algum tempo, os serviços *Web* eram sinónimo de SOAP. SOAP pode ser vista como uma implementação *Web* do conceito Remote Procedure Call (RPC), recorrendo ao XML como formato de empacotamento. Cedo se percebeu que apesar do SOAP usar as tecnologias *Web*, não abraçava os princípios arquiteturais da *Web*. [15], [16]

Algum tempo após o REST ter sido conceptualizado, tornou-se claro que o SOAP tinha um concorrente de peso que seguia melhor os princípios arquiteturais da *Web*. Hoje em dia, o tipo de serviços *Web* onde se enquadra o SOAP é referido como serviços *Web* "WS-*", espelhando a evolução que sofreram desde o surgimento do SOAP. Ainda assim, os serviços ou APIs *Web* RESTful, assim denominadas porque implementam o estilo arquitetural REST, ganharam popularidade devido às suas vantagens em termos de simplicidade, acoplamento, interoperabilidade e escalabilidade. Esta popularidade levou a que muitos serviços *Web*, hoje em dia, sejam promovidos como sendo RESTful apesar de não seguirem todos os princípios arquiteturais e restrições do estilo REST. [15], [16]

2.4.1 ESTILO ARQUITETURAL REST

Um dos trabalhos de investigação mais relevantes e influentes para a maneira como atualmente se desenvolvem sistemas baseados na *Web* foi feito por Roy Fielding. Em 2000, Roy Fielding, um dos contribuidores chave para o HTTP e URI, na sua tese de doutoramento intitulada *"Architectural Styles and Design of Network-based Software Architectures"* generalizou os princípios da arquitetura da *Web* e apresentou-os como um estilo arquitetural. Fielding definiu um estilo arquitetural como sendo um conjunto de restrições arquiteturais que restringem os papéis ou funcionalidades dos elementos arquiteturais e as relações permitidas entre esses elementos em qualquer arquitetura que obedeça a

esse estilo. Fielding também sugeriu que o espaço de todos os estilos arquiteturais possíveis pode ser visto como uma árvore de derivação, onde cada nó é derivado de outros através da adição de restrições. Alguns destes nós correspondem a estilos arquiteturais "básicos" bem conhecidos, enquanto que outros correspondem a combinações ou derivações dos estilos "básicos". Dado que cada estilo arquitetural induz um conjunto de propriedades arquiteturais, ao percorrer a árvore é possível perceber as propriedades que uma arquitetura específica, que adere a um estilo arquitetural, vai exibir assim que estiver implementada. [15]–[17]

Na mesma tese, Fielding descreveu o estilo arquitetural REST, que é derivado de um conjunto de estilos bem conhecidos, como por exemplo o estilo cliente-servidor. Em suma, REST é um estilo arquitetural, para aplicações de rede, que impõe um conjunto de restrições. Ao seguir estas restrições, é suposto que o resultado seja um sistema altamente escalável, com capacidade para crescer organicamente de uma maneira descentralizada. Esta alegação é comprovada pela *Web*, que é um sistema que implementa as restrições referidas, e por exemplos de componentes *Web* que violaram estas restrições e em consequência sofreram graves problemas, tais como problemas de escalabilidade. Seguidamente, são apresentadas as restrições que compõem o estilo arquitetural REST [17].

CLIENTE-SERVIDOR

A primeira restrição imposta é a utilização do estilo arquitetural cliente-servidor. Este estilo traduz-se, basicamente, num cliente que envia um pedido para o servidor e este envia uma resposta para o cliente. O princípio por detrás da restrição cliente-servidor é a separação de interesses. Ao separar os interesses da interface do utilizador, dos interesses do armazenamento dos dados, a portabilidade da interface do utilizador é melhorada e a escalabilidade também é melhorada devido à simplificação dos componentes do servidor. A separação de interesses também permite que os componentes evoluam separadamente.

AUSÊNCIA DE ESTADO

Esta restrição, adicionada à interação cliente-servidor, impõe que as comunicações sejam desprovidas de estado. Cada pedido efetuado do cliente para o servidor deve conter toda a informação necessária para o pedido ser entendido pelo servidor. Para além disto, os pedidos não podem tirar partido de qualquer contexto armazenado no servidor. Deste modo, o estado da sessão é mantido inteiramente no cliente. Esta restrição tem alguns impactos positivos na visibilidade, fiabilidade e escalabilidade. A visibilidade é melhorada porque não é necessário verificar mais que um pedido para se compreender inteiramente o pedido. A fiabilidade é melhorada porque a recuperação de falhas parciais é facilitada. A escalabilidade é melhorada porque, ao não ter que armazenar a sessão entre pedidos, o servidor pode libertar recursos rapidamente e também leva à simplificação da implementação porque o servidor não tem que gerir a utilização dos recursos entre pedidos. Esta restrição, para além das vantagens referidas anteriormente, também traz algumas desvantagens. Como o contexto não pode ser guardado no servidor, é necessário enviar todos os dados necessários em cada pedido, o que pode levar a que exista uma grande repetição de dados em cada pedido e assim diminuir o desempenho da rede. Adicionalmente, colocar o estado da aplicação no lado do cliente reduz o controlo do servidor sobre o comportamento da aplicação, dado que a aplicação torna-se dependente da correta implementação nas múltiplas versões do cliente.

CACHE

De modo a melhorar a eficiência da rede, foi adicionado restrições de *cache*. Através do mecanismo de *cache*, um cliente pode reutilizar os dados de uma resposta do servidor nos pedidos subsequentes que sejam equivalentes. A vantagem de adicionar restrições de *cache* está na possibilidade de poderem ser eliminadas algumas interações, melhorando a eficiência, escalabilidade e performance. A desvantagem é que a *cache* pode diminuir a fiabilidade se os dados presentes na *cache* diferirem substancialmente dos dados que seriam obtidos se o pedido fosse enviado diretamente para o servidor.

INTERFACE UNIFORME

A principal funcionalidade que distingue o estilo de arquitetura REST dos outros estilos baseados em rede é a ênfase dada a uma interface uniforme entre componentes. Todas as interações devem ser feitas através de uma interface uniforme que suporta as interações com os recursos através de um conjunto de métodos. Um serviço que implementa o estilo arquitetural REST expõe recursos, e as interações com um recurso só podem ser feitas através da interface uniforme. Uma interface uniforme implica a existência de algumas restrições arquiteturais que guiem o comportamento dos componentes. No caso do REST, existem quatro restrições de interface, que são descritas seguidamente.

Identificação de recursos. Normalmente, nos sistemas que se constroem, existe um conjunto de abstrações que importa atribuir uma identificação. Tudo o que deve ser identificável deve receber um identificador único e global de maneira a ser referenciado independentemente do contexto. Na *Web* existe um conceito unificador para a identificação de recursos: o URI. O uso de URIs para identificar os recursos chave significa que eles recebem um identificador único e global. Ao usar o esquema URI, colhe-se os benefícios de usar um esquema bem definido, que funciona bem à escala global e é compreendido por muitas pessoas. Neste contexto, um recurso tem o mesmo significado que no contexto do HTTP, que pode ser consultado na secção 2.3.2.

Manipulação de recursos através de representações. Os componentes REST executam ações nos recursos através do uso de representações que capturam o estado atual, ou desejado, do recurso. Tal como um recurso, neste contexto, uma representação tem o mesmo significado que no contexto do HTTP, que pode ser consultado na secção 2.3.3.

Mensagens auto descritivas. As mensagens trocadas entre os componentes têm que ser auto descritivas de modo a serem passíveis de ser processadas por intermediários (*proxies*, *gateways*). Uma mensagem ser auto descritiva significa que a mensagem deve conter toda a informação necessária para o recipiente a compreender. Existem alguns aspetos que contribuem para que uma mensagem seja auto descritiva. Ao garantir uma interação desprovida de estado entre o cliente e o servidor, o servidor pode compreender um pedido do cliente sem ter que aceder a outra informação armazenada no servidor que esteja relacionada com os pedidos anteriores. Adicionalmente, a utilização de métodos e tipos de média normalizados, leva a que os intervenientes saibam como interpretar e processar uma mensagem. Por fim, ao indicar explicitamente numa resposta, se esta pode ser usada para fins de *cache*, o cliente não tem que "adivinhar" se deve colocar a mensagem em *cache*, e o servidor não tem que transmitir esta informação fora da mensagem.

HATEOAS. A ideia por detrás do Hypermedia as the Engine of Application State (HATEOAS) é que os clientes devem interagir, e, possivelmente, alterar o estado da aplicação, através de controlos hipermédia contidos dentro das representações trocadas entre os servidores e os clientes. Esta ideia leva a que um cliente só necessite de ter conhecimento do ponto de entrada da aplicação e, através dos controlos hipermédia, consiga descobrir os recursos da aplicação e as ações que pode executar sobre estes. Esta ideia está bastante latente nas páginas *Web*, sendo que um cliente apenas necessita de saber o endereço da página, e, através dos controlos presentes na página, pode, por exemplo, navegar para outras páginas. Esta restrição é, provavelmente, a mais importante para suportar o desacoplamento entre um cliente e um servidor, porque os recursos podem ser descobertos em tempo de execução e pode-se interagir com eles através da interface uniforme sem a necessidade de entendimento prévio entre as partes intervenientes.

SISTEMA EM CAMADAS

Um sistema organizado em camadas impõe que cada componente de uma camada apenas tem conhecimento da camada com que interage. Esta restrição combinada com a restrição cliente-servidor implica que a adição de qualquer componente, como é o caso de um *proxy* ou de um *gateway*, entre o cliente e o servidor é uma operação quase transparente. Um cliente não sabe se está a comunicar diretamente com um servidor ou se está a comunicar com o servidor através de outros componentes.

CÓDIGO A PEDIDO

Ao permitir que as funcionalidades dos clientes sejam estendidas através do descarregamento e execução de código, na forma de *applets* ou *scripts*, os clientes são simplificados pois o número de funcionalidades pré-implementadas é diminuído. Ao permitir que as funcionalidades sejam descarregadas quando o sistema já está em produção, a extensibilidade do sistema é melhorada mas a visibilidade é reduzida. Por este motivo, esta restrição é opcional.

2.4.2 DOCUMENTAÇÃO DOS SERVIÇOS RESTFUL

Para usar um serviço *Web* é necessário ter conhecimento da sua existência e de como o usar. Os primeiros serviços *Web*, baseados em SOAP, recorreram à ideia de um diretório de serviços para localizar os serviços, usando para isso o Universal Description, Discovery and Integration (UDDI), conjuntamente com o Web Services Description Language (WSDL) para descrever os serviços. Em serviços baseados em SOAP, cada serviço expõe uma interface específica, tornando-se necessário uma descrição da interface para se poder usar o serviço. Deste modo, a descrição e localização de serviços, tornou-se um ponto importante no desenvolvimento de serviços *Web* até ao surgimento dos serviços RESTful. Nos serviços RESTful, devido à interface uniforme e à restrição HATEOAS imposta pelo REST, não existe a necessidade de uma descrição específica da interface de um serviço, nem a necessidade de usar um mecanismo específico para localizar os serviços. Ainda assim, apesar de não ser estritamente necessário, a descrição de serviços RESTful pode ser útil. Uma das razões para essa utilidade prende-se com a necessidade de fornecer documentação para os utilizadores de um serviço. Deve-se ter em conta o risco de a documentação ficar desatualizada com novas versões do serviço,

e por isso os clientes do serviço devem sempre basear-se nas funcionalidades oferecidas pelo REST (interface uniforme, negociação de conteúdo e HATEOAS). Não obstante, surgiram várias propostas para descobrir, descrever e perceber o significado de um serviço RESTful.

A descrição de um serviço passa por descrever os recursos, os URLs, os formatos de representação dos recursos (JSON, XML, etc.), os códigos de estado e os possíveis argumentos. Trata-se de uma área que tem tido bastante ênfase no desenvolvimento de uma API. Uma das primeiras propostas a ganhar popularidade foi o Web Application Description Language (WADL), que é visto como sendo o equivalente do WSDL. O ponto fraco do WADL é a falta de suporte para a natureza hipermédia dos serviços RESTful. Atualmente, os formatos mais conhecidos para descrever um serviço são o Swagger, RESTful API Modeling Language (RAML) e API Blueprint [18]–[20]. O modo de funcionamento destas propostas é semelhante e passa por descrever a API RESTful recorrendo a uma meta-linguagem baseada em XML, JSON ou YAML Ain't Markup Language (YAML), resultando num conjunto de documentos que podem ser usados para gerar documentação para o cliente da API ou mesmo para gerar uma aplicação cliente da API.

Descobrir um serviço consiste em procurar e selecionar o serviço que oferece as funcionalidades desejadas de acordo com os critérios estabelecidos (preço, desempenho, etc.). Atualmente, existe um conjunto de serviços que funcionam como motor de pesquisa de serviços *Web*. Um dos exemplos mais conhecidos é o ProgrammableWeb API Directory [21]. Outro exemplo é o APIs.io [22], que tem a particularidade de usar o formato APIs.json [23] para encontrar e listar os serviços *Web*.

Perceber o significado de um serviço consiste em conhecer a semântica, para além da que é definida pelos tipos de média, associada aos recursos. Esta semântica, ou vocabulário de termos que são transportados nos pedidos e respostas, é definida através de perfis. Apesar de ainda ser uma especificação instável e com pouco uso, o Application-Level Profile Semantics (ALPS) apresenta diversas vantagens em relação a outras abordagens, existentes atualmente, para descrever a semântica ao nível da aplicação. Entre essas vantagens encontra-se a simplicidade e capacidade de reutilização dos perfis [24], [25].

2.4.3 TIPOS DE HIPERMÉDIA

Como Mike Amundsen refere, "um tipo de hipermédia é um tipo de média que contém elementos nativos de hiperligação que podem ser usados para controlar o fluxo da aplicação" [26]. É através de um tipo de hipermédia que se pode construir um serviço RESTful que obedeça à restrição HATEOAS. Um exemplo de tipo de hipermédia bastante conhecido é o HTML. O sucesso do HTML, como tipo de média, na *Web*, deve-se em grande parte ao seu suporte nativo de um conjunto alargado de controlos hipermédia, ao contrário do XML ou JSON. O sucesso do HTML não se alastrou para os serviços RESTful, deixando um espaço por preencher no que concerne aos tipos de hipermédia. Para contornar a inexistência de controlos hipermédia nos tipos de média mais usados, como são os casos do JSON e XML, foram criados tipos de hipermédia que estendem estes para incluir controlos hipermédia. São exemplos destes tipos de hipermédia o Collection+JSON, o Hypertext Application Language (HAL), o JSON for Linking Data (JSON-LD) e o Structured Interface for Representing Entities (SIREN). Apesar de não existirem números concretos para afirmar que o HAL é o tipo de hipermédia mais popular no contexto dos serviços RESTful, é seguro dizer que o HAL tem ganho popularidade ao ponto de ter sido adotado pela Amazon [27], [28].

O HAL é um tipo de hipermédia que prima pela simplicidade. O HAL impõe o mínimo de

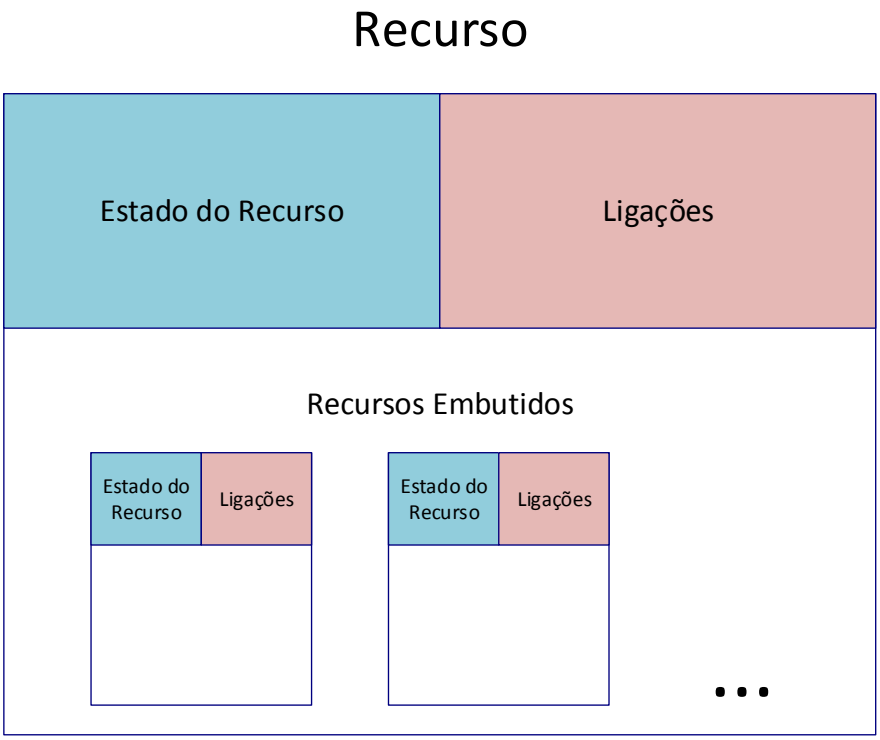

Figura 2.6: Estrutura de uma representação HAL

estrutura necessária para satisfazer a restrição HATEOAS. O HAL tem uma versão que usa JSON (application/hal+json) e uma versão que usa XML (application/hal+xml). Independentemente da versão, o HAL assenta em dois conceitos: recursos e ligações. Um recurso pode ser composto por ligações para URIs, por recursos embutidos e pelo estado do recurso. Uma ligação pode ser composta por um alvo, uma relação e propriedades opcionais. A figura 2.6 ilustra um recurso no tipo de hipermédia HAL.

A listagem 1 mostra um exemplo simples de um recurso no formato HAL+JSON que modela um colaborador de uma empresa. O colaborador possui duas ligações, uma com a relação "self" que aponta para o próprio colaborador e uma com a relação "departamento" que aponta para o departamento a que o colaborador pertence. Além das ligações, o recurso possui o estado do colaborador (nome e numero).

```
{
    "_links": {
        "self": {
            "href": "/colaboradores/123"
        },
        "departamento": {
            "href": "/departamentos/5"
        }
    },
    "nome": "José",
    "numero": 123
}
```

Listagem 1: Exemplo HAL+JSON

2.4.4 MODELO DE MATURIDADE DE RICHARDSON

No panorama atual dos serviços *Web*, existem muitos serviços que se proclamam RESTful apesar de não seguirem todos os princípios arquiteturais e restrições do estilo REST. Roy Fielding endereçou esta preocupação, referindo algumas regras que devem ser seguidas, e que são constantemente violadas, no desenho de uma API antes desta poder ser classificada como sendo RESTful [29]. Um modelo popular para classificar os serviços *Web* foi desenvolvido e apresentado, na conferência QCon em 2008, por Leonard Richardson. Richardson, para desenvolver este modelo, teve como motivações o desejo de desenhar bons serviços *Web*, encontrar uma maneira de distinguir bons serviços *Web* de maus serviços *Web* e descobrir a relação existente entre as restrições do estilo arquitetural REST e a qualidade dos serviços *Web*. O modelo de Richardson classifica a maturidade de um serviço em quatro níveis tendo como base a forma como o serviço lida com URIs, HTTP e hipermédia. [30]–[32]

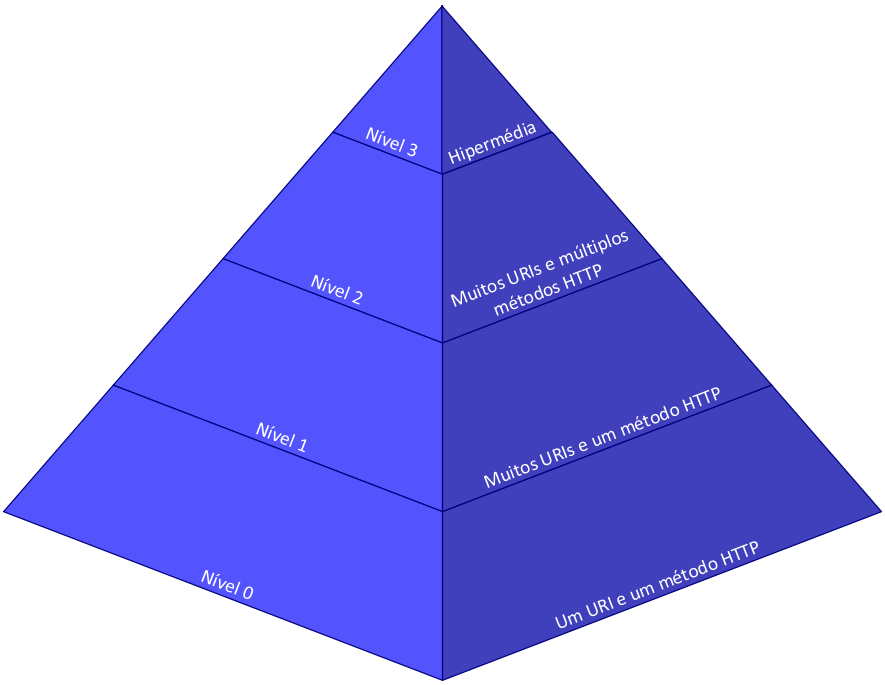

Figura 2.7: Modelo de maturidade de Richardson

NÍVEL 0

O nível 0, o mais básico do modelo, representa os serviços que têm apenas um URI e que usam somente um método HTTP. Os serviços SOAP e os serviços XML-RPC são exemplos de serviços que se enquadram neste nível. Ambos os tipos de serviços disponibilizam unicamente um URI para onde são feitos todos os pedidos HTTP, que são caracterizados por serem todos do tipo POST, não sendo usado o resto dos métodos disponíveis no protocolo HTTP.

NÍVEL 1

Os serviços do nível 1, tal como os do nível 0, fazem uso de apenas um dos métodos HTTP mas ao contrário dos serviços do nível 0, os serviços deste nível disponibilizam vários URIs para aceder aos recursos expostos. Deste modo, em vez de se fazer todos os pedidos para um único URI, que representa um recurso, os pedidos são feitos para os vários recursos disponíveis, usando para isso diferentes URIs. Muitos dos serviços que se autoproclamam RESTful são, na realidade, serviços de nível 1. São exemplos disto, os serviços em que as ações que se podem efetuar sobre um recurso estão mapeadas nos URIs, não tirando partido da expressividade do HTTP.

NÍVEL 2

Tal como o nível 1, os serviços do nível 2 disponibilizam vários URIs mas cada URI suporta múltiplos métodos HTTP. Trata-se de serviços que usam devidamente os métodos HTTP.

NÍVEL 3

No nível 3 é introduzido o HATEOAS, permitindo que os serviços descrevam as ações que se podem realizar e os recursos que são necessários manipular para concretizar essas ações. Deste modo, os clientes podem explorar a panóplia de recursos disponibilizados em vez de terem que saber de antemão tudo o que o serviço oferece.

capítulo 3

Requisitos da API

Este capítulo é dedicado, em grande parte, à análise dos requisitos funcionais e não-funcionais da API. Para auxiliar na compreensão dos requisitos, antecedentemente são apresentados os conceitos do domínio do problema.

3.1 visão geral

Como foi explicado no capítulo 2, o PM é um aplicação de gestão de desempenho que tem duas versões, uma que corre no *cluster* regional (NetAct PM) e outra que corre no *cluster* global (NPM). No caso do NetAct PM, foi criada uma API para responder às necessidades das outras aplicações existentes no NetAct do *cluster* regional. Pode-se dar o exemplo da aplicação Optimizer, que usa as estatísticas de desempenho fornecidas pela API do PM para oferecer ferramentas que permitam otimizar o desempenho e utilização de recursos da rede de acesso rádio, de uma maneira rápida e precisa. Esta API oferece um subconjunto das funcionalidades disponíveis através do portal *Web* do PM. Atualmente, a API implementada já não consegue responder às necessidades dos clientes, apresentando diversos problemas, entre os quais problemas de desempenho. Conjuntamente, devido à inexistência de um conjunto alargado de testes automáticos de software e à qualidade do código da API, as tarefas de manutenção e extensão da API, tendo em vista a inclusão de novas funcionalidades, tornaram-se bastante difíceis. Fazendo uma análise a esta API, de acordo com o modelo de maturidade de Richardson, apresentado no capítulo 2, chega-se à conclusão que, também neste campo, a API não colhe bons resultados, sendo classificada como uma API RESTful de nível 1, pois não usa devidamente os métodos HTTP e também não suporta o HATEOAS.

Decorrente de tudo o que foi referido anteriormente e do surgimento de novos requisitos por parte dos clientes, chegou-se à conclusão que seria uma melhor opção desenvolver uma nova API RESTful que implementasse as boas práticas de desenvolvimento de código e que estivesse em linha com o que de melhor se faz no campo das APIs RESTful. Com esta API pretende-se suportar não só os atuais clientes, que são as aplicações existentes no NetAct do *cluster* regional, como também possíveis novos clientes, que podem ir desde as aplicações existentes no *cluster* global até uma aplicação iOS que se encontra em desenvolvimento com o intuito de oferecer algumas das funcionalidades que estão disponíveis através do portal *Web* do PM. Este portal *Web* do PM, que se encontra em processo de

migração de Adobe Flex para HTML 5, é precisamente um dos novos clientes da API. Assim, a API não será usada apenas por equipas de desenvolvimento externas ao PM, como também será usada pela própria equipa de desenvolvimento do PM.

3.2 CONCEITOS DO PROBLEMA

Para se entender os requisitos da API importa perceber os conceitos do domínio do problema. Imagine-se uma rede de telecomunicações móveis constituída por vários elementos de rede e pelo PM, tal como mostra a figura 3.1. Na terminologia do PM, os elementos de rede são objetos que formam a topologia da rede. Estes objetos podem constituir hierarquias com diversos níveis, como por exemplo a hierarquia (elemento de rede α)/(elemento de rede β), mostrada na figura 3.1, que tem 2 níveis. Cada objeto insere-se numa classe que classifica o objeto. Os elementos de rede enviam estatísticas para o PM, como por exemplo o número de chamadas efetuadas em cada hora ou o número de chamadas perdidas em cada hora. Estas estatísticas são denominadas de contadores ou indicadores de desempenho e têm valores associados. Como exemplo, o elemento de rede α pode enviar que entre as 14:00 e as 15:00 foram efetuadas 40 chamadas. Ambos os contadores (número de chamadas efetuadas e perdidas) podem ser categorizados como sendo "estatísticas de tráfego na rede". Esta categorização é o que se denomina no PM por medida. Uma medida é uma coleção de contadores logicamente relacionados. Os elementos de rede enviam para o PM as medidas sob a forma de um relatório que contém o nome e o período da medida, os contadores e seus valores e o objeto a que a medida diz respeito. Todos estes relatórios são dados em bruto que podem ser posteriormente processados pelo PM de modo a gerar outro relatório. Continuando com o exemplo que tem sido dado, se o elemento de rede β enviar relatórios de hora a hora com o número de chamadas efetuadas, então envia 24 relatórios por dia. O PM pode pegar nestes dados em bruto e gerar um relatório com a evolução, ao longo do dia, do número de chamadas efetuadas no elemento de rede β. Através deste relatório é possível saber qual a hora mais movimentada no elemento de rede β, ou alternativamente, pode-se aplicar uma fórmula (máximo) nos dados em bruto para extrair esta informação. Existem outros valores relevantes que podem ser calculados a partir dos dados, como por exemplo a percentagem de chamadas efetuadas que foram perdidas. Este tipo de fórmulas são os denominados indicadores chave de desempenho (KPI - Key Performance Indicator). Sobre estes KPIs, ou sobre os contadores, podem ser definidos limiares que quando são alcançados emitem um alarme. Deste modo, é garantido que os problemas de desempenho são identificados atempadamente.

Através do PM, também se pode querer saber como é que a rede se está a comportar, abstraindo-se dos detalhes específicos de cada elemento de rede. Isto é alcançado através da combinação dos dados em bruto de todos os elementos de rede. No exemplo dado, isto significa que os dados resultantes contêm o número de chamadas efetuadas em toda a rede. Define-se isto como sendo uma agregação dimensional de objetos. Como todos os elementos de rede enviam $24 * 365$ relatórios por ano, para poupar espaço em disco, pode ser feito um cálculo para obter o valor semanal de chamadas efetuadas para cada elemento de rede e assim em vez de terem que ser guardados $24 * 7$ relatórios por semana e por elemento de rede, apenas é necessário guardar 1 relatório por elemento de rede para cada semana. Isto é um exemplo de uma agregação em que a dimensão é o tempo.

Um dos conceitos ainda não referidos nesta secção e que está presente, de alguma maneira, em praticamente todas as funcionalidades é a adaptação. Tal como foi descrito na secção 2.1.3 do capítulo

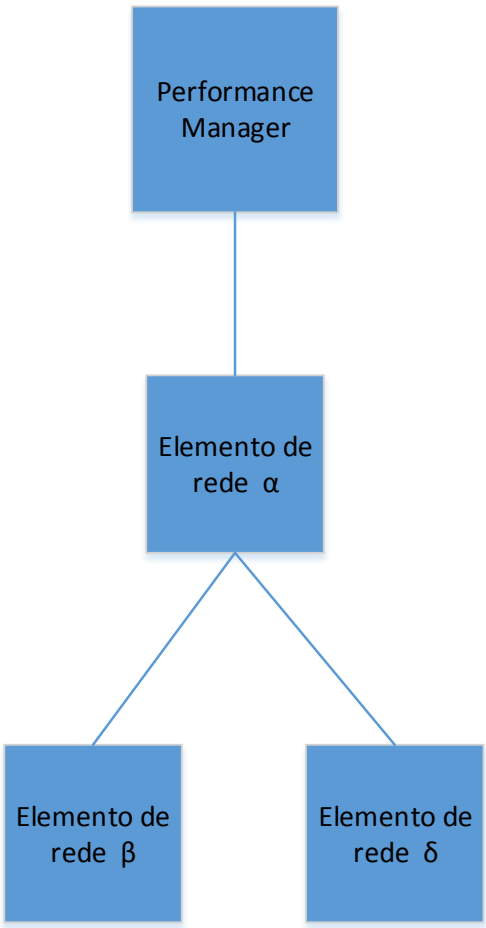

Figura 3.1: Exemplo de integração do PM com a rede

2, uma adaptação define como é que se deve interpretar e guardar os dados dos elementos de rede. Os contadores, medidas, KPIs e relatórios pré-configurados são disponibilizados através de pacotes de conteúdo e cada pacote de conteúdo tem uma versão associada.

Pode-se representar visualmente os conceitos através de um modelo de conceitos. O modelo de conceitos apresentado na figura 3.2 apenas abrange os conceitos relevantes para os casos de uso que são descritos na secção 3.3.

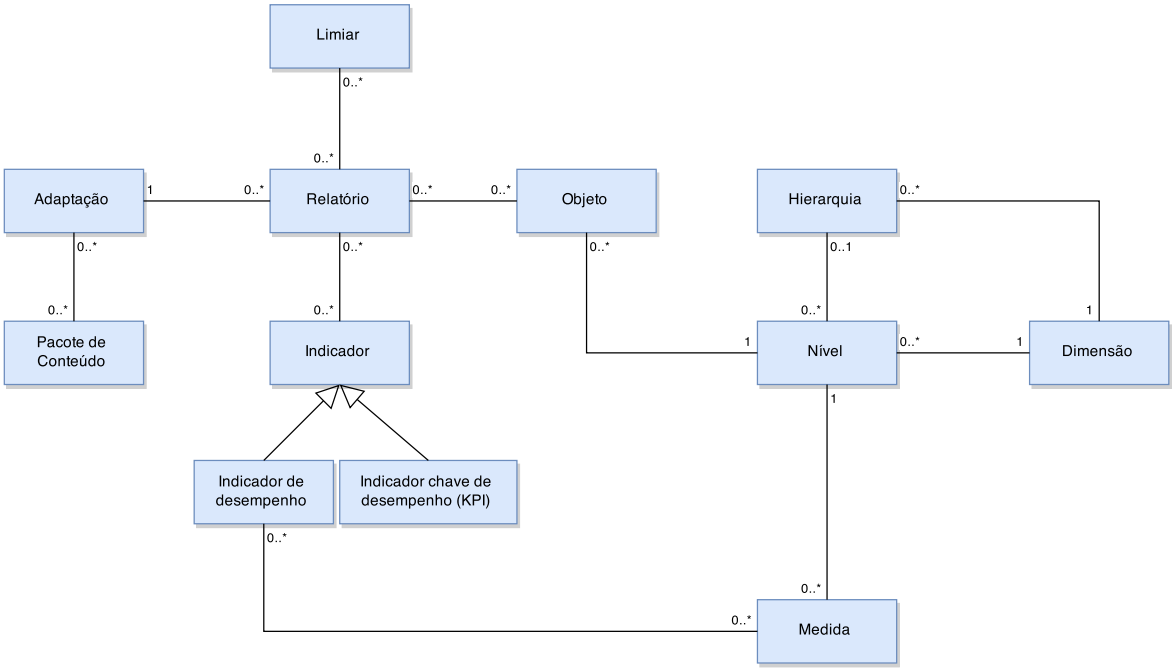

Figura 3.2: Modelo de Conceitos

3.3 CASOS DE USO

Da análise à API existente e às novas funcionalidades requisitadas, resultaram os casos de uso que são apresentados no diagrama de casos de uso da figura 3.3 e que são descritos seguidamente.

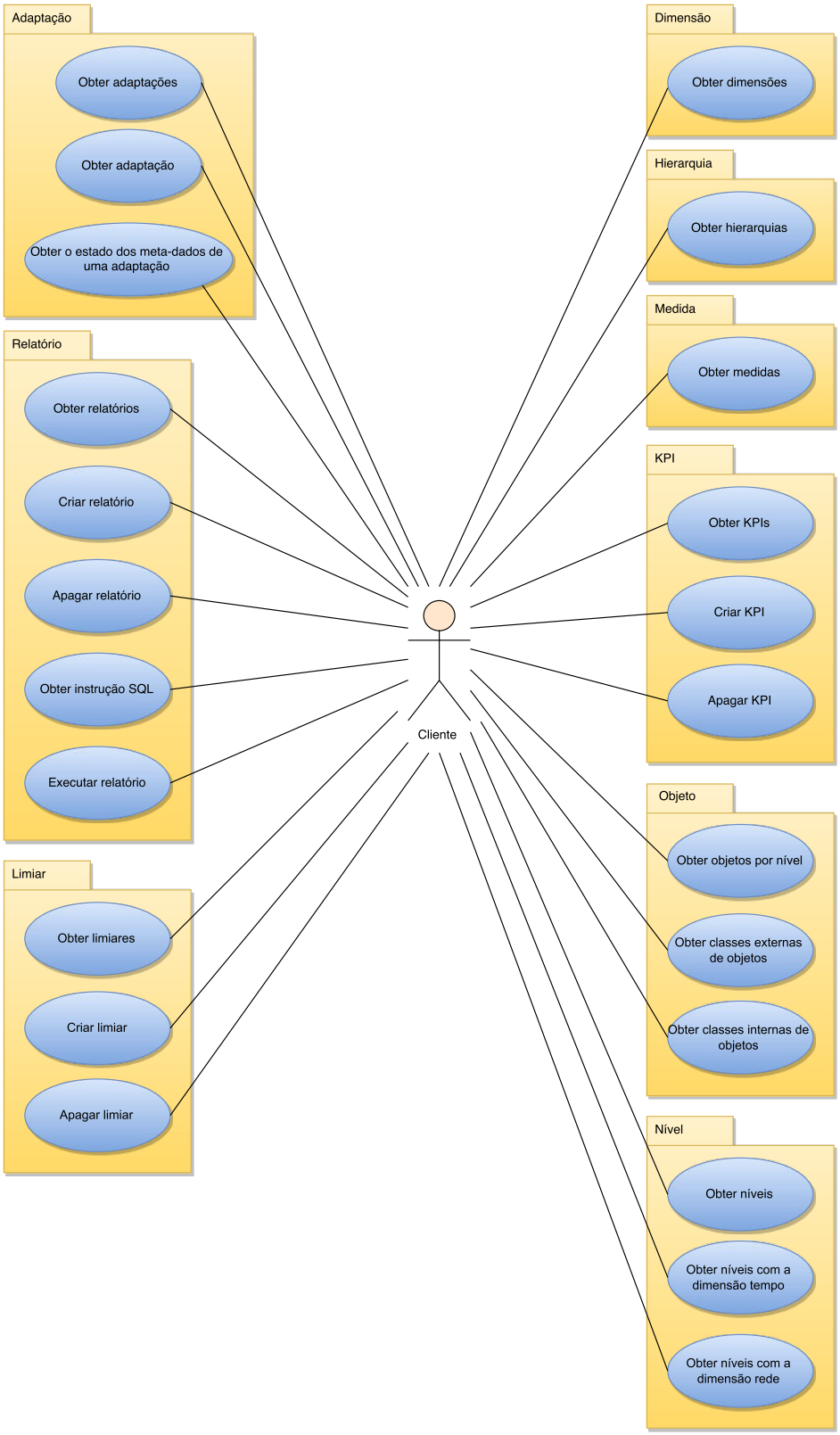

Figura 3.3: Diagrama de Casos de Uso

3.3.1 OBTER ADAPTAÇÕES

Um cliente acede à API e faz um pedido para obter todas as adaptações. A API responde com uma listagem de todas as adaptações instaladas no sistema. Caso o número de adaptações instaladas seja elevado, a API não as lista todas de uma só vez. A API também deve listar as adaptações combinadas.

3.3.2 OBTER ADAPTAÇÃO

Um cliente acede à API e faz um pedido para obter os detalhes de uma adaptação. Neste pedido o cliente identifica qual é a adaptação desejada. Caso a adaptação esteja instalada no sistema, a API responde com os detalhes da adaptação. Caso contrário, a API notifica o cliente que a adaptação indicada é inválida.

3.3.3 OBTER O ESTADO DOS META-DADOS DE UMA ADAPTAÇÃO

Um cliente acede à API e faz um pedido para verificar se o pacote base de conteúdo a que uma adaptação está associada foi atualizado desde uma determinada altura. Na resposta, a API indica se o pacote base de conteúdo foi atualizado ou não. No pedido, o cliente tem que identificar a adaptação pretendida e tem a opção de indicar um instante de tempo. Caso este instante de tempo seja fornecido então a API vai verificar se o pacote base da adaptação foi atualizado desde esse instante. Caso não seja fornecido nenhum instante de tempo então a API vai verificar se o pacote base da adaptação sofreu alguma atualização desde que foi instalado no sistema. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida.

3.3.4 OBTER DIMENSÕES

Um cliente acede à API e faz um pedido para obter todas as dimensões (dimensão rede e dimensão tempo) que se podem relacionar com uma certa adaptação identificada no pedido. O cliente também tem a possibilidade de indicar as versões dos pacotes de conteúdo para os quais quer resultados. Caso o cliente não refira nenhuma versão do pacote de conteúdo, então a API retorna resultados para todas as versões dos pacotes de conteúdo a que a adaptação indicada está associada. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se alguma das versões dos pacotes de conteúdo indicadas também não existir no sistema ou existir mas não estar associada à adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida.

3.3.5 OBTER HIERARQUIAS

Um cliente acede à API e faz um pedido para obter todas as hierarquias de uma dada adaptação indicada. Também é necessário indicar a versão do pacote de conteúdo desejado. Opcionalmente, o cliente também pode indicar para que dimensão é que pretende as hierarquias. Por omissão são

retornados resultados para a dimensão rede. Caso o número de hierarquias encontradas seja elevado, a API não as deve fornecer todas de uma só vez. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se a versão do pacote de conteúdo indicada também não existir no sistema ou existir mas não estar associada à adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar uma dimensão mas esta não existir, a API notifica o cliente que a dimensão é inválida.

3.3.6 OBTER RELATÓRIOS

Um cliente acede à API e faz um pedido para obter todos os relatórios para uma adaptação e versões de pacotes de conteúdo indicados. Opcionalmente, o cliente pode indicar que quer relatórios só de um determinado utilizador. Por omissão são retornados os relatórios privados do utilizador que se autenticou e fez o pedido, os relatórios partilhados por outros utilizadores e os relatórios padrão. Caso o cliente que faz o pedido seja administrador do sistema e indicar outro utilizador que não ele próprio, então a API retorna os relatórios privados do utilizador indicado, os relatórios partilhados e os relatórios padrão. Se um utilizador que não tem privilégios de administrador indicar outro utilizador no pedido que não o próprio, a API notifica que o utilizador não está autorizado a aceder a relatórios de outros utilizadores. Além de poder indicar o utilizador, também existe a possibilidade de o cliente indicar os níveis para os quais quer os relatórios. Por omissão são retornados relatórios para todos os níveis. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se alguma das versões dos pacotes de conteúdo indicada também não existir no sistema ou existir mas não estar associada à adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar um utilizador não existente, a API notifica o cliente que o utilizador é inválido. Caso o número de relatórios encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.7 CRIAR RELATÓRIO

Um cliente acede à API e faz um pedido para criar um relatório. Neste pedido o cliente deve fornecer todos os dados necessários para criar o relatório. Isto inclui fornecer um nome para o relatório, a adaptação, versões de pacotes de conteúdo, indicadores (KPIs, contadores, etc.), níveis para as várias dimensões e o período de tempo para o qual se deseja ter dados resultantes da execução do relatório. Caso o relatório não seja criado com sucesso, a API informa o cliente de qual é a causa do problema.

3.3.8 APAGAR RELATÓRIO

Um cliente acede à API e faz um pedido para apagar um relatório. O cliente tem que identificar devidamente qual o relatório que pretende apagar. Um cliente com privilégios de administrador pode apagar qualquer relatório do sistema. Um cliente sem privilégios de administrador apenas pode apagar os relatórios que a si pertencem e caso tente apagar um relatório de outro utilizador, a API notifica o cliente que ele não tem privilégios para apagar relatórios de outros utilizadores. Se o cliente indicar um relatório inexistente no sistema, a API notifica o cliente que não foi possível apagar o relatório

pois ele não existe.

3.3.9 OBTER INSTRUÇÃO SQL

Um cliente acede à API e faz um pedido para obter a instrução Structured Query Language (SQL) que gera os dados para um relatório indicado. O cliente pode fornecer um relatório que não exista no sistema, desde que este esteja de acordo com o formato esperado. Caso o relatório definido pelo cliente apresente irregularidades, a API notifica o cliente que o relatório é inválido.

3.3.10 EXECUTAR RELATÓRIO

Um cliente acede à API e faz um pedido para obter os dados resultantes da execução de um relatório fornecido. O cliente pode fornecer um relatório que não exista no sistema, desde que este esteja de acordo com o formato esperado. Caso o relatório definido pelo cliente apresente irregularidades, a API notifica o cliente que o relatório é inválido. Deve-se ter em conta o tempo de execução de um relatório que pode ser elevado, tal como a quantidade de dados resultantes da execução. Estas preocupações advém das limitações da API atual que têm um grande impacto nos clientes.

3.3.11 OBTER MEDIDAS

Um cliente acede à API e faz um pedido para obter todas as medidas para uma adaptação e versão de pacote de conteúdo indicadas. Opcionalmente, o cliente pode indicar os níveis para os quais quer as medidas. Por omissão, a API retorna medidas para todos os níveis. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se a versão do pacote de conteúdo indicada também não existir no sistema ou existir mas não estar associada com a adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar algum nível inexistente, a API notifica o cliente que o nível é inválido. Caso o número de medida encontradas seja elevado, a API não as deve retornar todos de uma só vez para o cliente.

3.3.12 OBTER KPIS

Um cliente acede à API e faz um pedido para obter todos os KPIs válidos para uma adaptação e versão de pacote de conteúdo indicadas. Opcionalmente, o cliente pode indicar os níveis e o utilizador para os quais quer os KPIs. Caso não seja indicado nenhum nível, a API retorna KPIs para todos os níveis. Caso não seja indicado nenhum utilizador, a API retorna os KPIs privados do utilizador e os KPIs publicados. Caso o cliente que faz o pedido seja administrador do sistema e indicar outro utilizador que não ele próprio, então a API retorna os KPIs privados do utilizador indicado e os KPIs publicados. Se um utilizador que não tem privilégios de administrador indicar outro utilizador no pedido que não o próprio, a API notifica que o utilizador não está autorizado a aceder a KPIs de outros utilizadores. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se a versão do pacote de conteúdo indicada também não existir no

sistema ou existir mas não estar associada com a adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar algum nível inexistente, a API notifica o cliente que o nível é inválido. Caso o número de KPIs encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.13 CRIAR KPI

Um cliente acede à API e faz um pedido para criar um KPI. Neste pedido o cliente deve fornecer todos os dados necessários para criar o KPI. Quando o KPI é criado, fica como privado e apenas pode ser acedido pelo cliente que o criou ou pelo administrador. Caso o KPI não seja criado com sucesso, a API informa o cliente de qual é a causa do problema.

3.3.14 APAGAR KPI

Um cliente acede à API e faz um pedido para apagar um KPI. O cliente tem que identificar devidamente qual o KPI que pretende apagar. Um cliente com privilégios de administrador pode apagar qualquer KPI do sistema. Um cliente sem privilégios de administrador apenas pode apagar os KPIs que a si pertence e caso tente apagar um KPI de outro utilizador, a API notifica o cliente que ele não tem privilégios para apagar KPIs de outros utilizadores. Se o cliente indicar um KPI inexistente no sistema, a API notifica o cliente que não foi possível apagar o KPI pois ele não existe.

3.3.15 OBTER NÍVEIS

Um cliente acede à API e faz um pedido para obter todos os níveis suportados para uma adaptação e versão de pacote de conteúdo indicadas. Opcionalmente, o cliente pode indicar a dimensão para a qual quer os níveis. Por omissão, a API retorna níveis para a dimensão rede. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se a versão do pacote de conteúdo indicada também não existir no sistema ou existir mas não estar associada com a adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar uma dimensão mas esta não existir, a API notifica o cliente que a dimensão é inválida. Caso o número de níveis encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.16 OBTER NÍVEIS COM A DIMENSÃO TEMPO

Um cliente acede à API e faz um pedido para obter todos os níveis com a dimensão tempo para uma adaptação indicada. Opcionalmente, o cliente pode indicar as versões dos pacotes de conteúdo, um nível de rede e um conjunto de indicadores (KPIs, contadores, etc). Caso o cliente não refira nenhuma versão do pacote de conteúdo, então a API retorna resultados para todas as versões dos pacotes de conteúdo a que a adaptação indicada está associada. Por omissão, a API retorna resultados para todos os níveis de rede e indicadores. Se a adaptação indicada pelo cliente não estiver instalada no

sistema, a API notifica o cliente que a adaptação é inválida. Se for indicada alguma versão do pacote de conteúdo que não exista no sistema ou exista mas não esteja associada com a adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se o cliente indicar um nível de rede mas este não existir, a API notifica o cliente que o nível de rede é inválido. Se no pedido for referido algum indicador inexistente, a API notifica o cliente que o indicador é inválido. Caso o número de níveis encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.17 OBTER NÍVEIS COM A DIMENSÃO REDE

Um cliente acede à API e faz um pedido para obter todos os níveis com a dimensão rede para uma adaptação e versão de pacote de conteúdo indicadas. Opcionalmente, o cliente pode referir um conjunto de indicadores (KPIs, contadores, etc). Por omissão, a API retorna resultados para todos os indicadores. Se a adaptação indicada pelo cliente não estiver instalada no sistema, a API notifica o cliente que a adaptação é inválida. Se for indicada uma versão do pacote de conteúdo que não exista no sistema ou exista mas não esteja associada com a adaptação indicada, a API notifica o cliente que a versão do pacote de conteúdo é inválida. Se no pedido for referido algum indicador inexistente, a API notifica o cliente que o indicador é inválido. Caso o número de níveis encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.18 OBTER OBJETOS POR NÍVEL

Um cliente acede à API e faz um pedido para obter todos os objetos para um determinado nível. Adicionalmente, o cliente pode fornecer uma expressão para filtrar os objetos retornados pela API. Caso este filtro seja omitido, a API retorna todos os objetos para o nível indicado. Se o nível indicado pelo cliente não existir, a API notifica o cliente que o nível é inválido. Caso o número de objetos encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.19 OBTER CLASSES DE OBJETOS EXTERNAS

Um cliente acede à API e faz um pedido para obter as classes de objetos externas correspondentes às classes de objetos internas fornecidas.

3.3.20 OBTER CLASSES DE OBJETOS INTERNAS

Um cliente acede à API e faz um pedido para obter as classes de objetos internas correspondentes às classes de objetos externas fornecidas.

3.3.21 OBTER LIMIARES

Um cliente acede à API e faz um pedido para obter todos os limiares associados a um determinado relatório. Se o relatório indicado pelo cliente não existir no sistema, a API notifica o cliente que o relatório é inválido. Caso o número de limiares encontrados seja elevado, a API não os deve retornar todos de uma só vez para o cliente.

3.3.22 CRIAR LIMIAR

Um cliente acede à API e faz um pedido para criar um limiar. Neste pedido o cliente deve fornecer todos os dados necessários para criar o limiar, entre os quais se encontra o relatório ao qual se quer associar o limiar. Caso o limiar não seja criado com sucesso, a API informa o cliente de qual é a causa do problema.

3.3.23 APAGAR LIMIAR

Um cliente acede à API e faz um pedido para apagar um limiar. O cliente tem que identificar devidamente qual o limiar que pretende apagar. Se o cliente indicar um limiar inexistente no sistema, a API notifica o cliente que não foi possível apagar o limiar pois ele não existe.

3.4 ATRIBUTOS DE QUALIDADE

3.4.1 INTEGRIDADE

A API deve verificar que o formato dos dados que recebe está de acordo com o esperado. Isto consiste, entre outras coisas, em verificar que os pedidos enviados pelos clientes possuem todos os campos obrigatórios para que os pedidos sejam executados com sucesso. Outro exemplo é a verificação do formato das datas, que, de modo a evitar confusão na sua interpretação, devem seguir uma notação pré-estabelecida e, de preferência, normalizada.

3.4.2 DESEMPENHO

Um cliente não deve ficar bloqueado por muito tempo à espera de uma resposta da API. Tipicamente, uma resposta HTTP tem um tempo limite de 30 segundos, o que já é considerado demasiado tempo de espera para a maior parte das interações. Face a isto, um limite razoável para o tempo de resposta da API será 10 segundos, o que se traduz em que qualquer interação com a API não deve demorar mais que 10 segundos.

3.4.3 SEGURANÇA

Para aceder às funcionalidades da API, os clientes têm que se autenticar e só devem conseguir realizar ações e aceder a informação para as quais estão autorizados. Pode-se dar como exemplo a funcionalidade de apagar relatórios em que só um cliente que se autentique como administrador é que pode apagar relatórios de outros utilizadores. Adicionalmente, as comunicações entre a API e os clientes devem ser cifradas.

3.4.4 MODIFICAÇÃO

A API deve ser desenhada e implementada de modo a precaver possíveis alterações ou inclusão de novas funcionalidades. Como exemplos, a API deve estar preparada para possíveis alterações das estrutura dos dados que são trocados entre a API e os clientes e também deve estar preparada para possíveis alterações dos URLs que são usados para fazer pedidos à API.

3.5 OUTROS REQUISITOS

3.5.1 QUANTIDADE DE DEFEITOS

A Nokia tem feito uma grande aposta na qualidade do software produzido dentro de portas e para suportar esta aposta definiu um conjunto de medidas que devem ser seguidas. Entre estas medidas, encontra-se uma que estabelece o número máximo de defeitos que um produto pode ter quando é disponibilizado para projetos-piloto em clientes selecionados e, numa fase seguinte, quando é entregue a versão final aos clientes. Nestas fases, o produto tem que ser entregue com zero defeitos críticos ou importantes e com um máximo de quinze defeitos menores. Como se insere no produto NetAct, a API não pode contribuir com defeitos que ponham em causa o cumprimento, por parte do NetAct, do que foi referido anteriormente. Isto leva a que a API não pode ter nenhum defeito critico ou importante e o mínimo possível de defeitos menores nas fases anteriormente referidas.

3.5.2 TESTES

Além da medida referida na secção 3.5.1, existem mais algumas medidas relacionadas com a qualidade que têm impacto no desenvolvimento da API, como é o caso da utilização de testes. Assim, todo o novo código desenvolvido no âmbito do NetAct deve ter testes de unidade e testes de aceitação que devem cobrir 90% do código. Para além disto, pelo menos 90% dos testes de aceitação devem ser automatizados.

3.6 RESTRIÇÕES

3.6.1 COMPATIBILIDADE

A API deve ser retrocompatível com as duas versões anteriores do NetAct PM. Quer isto dizer que a nova API tem que coexistir com a API antiga de maneira a que os clientes possam continuar a usar a API antiga como usavam até aqui e, caso pretendam, possam migrar para a nova API.

3.6.2 SERVIDOR APLICACIONAL

O PM, em produção, é instalado no servidor aplicacional WebSphere [33], o que acarreta que se tenha que usar a linguagem de programação Java ou outra linguagem que produza *software* que corra na máquina virtual Java. Adicionalmente, a versão do servidor aplicacional WebSphere usada atualmente só suporta até ao Java SE 5 e Java EE 1.4.

CAPÍTULO 4

Desenho da API

Neste capítulo são apresentados diversos aspetos que foram importantes no desenho da API, tendo por base os requisitos descritos no capítulo 3. Um dos pontos mais importantes no desenho de uma API RESTful é a modelação dos recursos e das relações entre eles. Precisamente por isso, neste capítulo é dada grande atenção aos recursos.

4.1 Recursos da API

Decorrente da análise aos casos de uso, foram identificados diversos recursos que a API tem que disponibilizar aos clientes. Este trabalho está focado no desenvolvimento de uma nova versão (segunda versão) da API do PM. Por esta razão, na análise que é feita aos recursos, não é dada relevância à versão existente (primeira versão) da API. Os recursos que a API tem que disponibilizar são:

(1) Coleção de Versões da API
  (I) Versão 2 da API
    (A) Coleção de Adaptações
      (i) Adaptação
        (a) Estado dos meta-dados
        (b) Coleção de Pacotes de conteúdo
          (1) Pacote de conteúdo
        (c) Coleção de Dimensões
          (1) Dimensão
        (d) Coleção de Hierarquias
          (1) Hierarquia
        (e) Coleção de Medidas
          (1) Medida
        (f) Coleção de Indicadores chave de desempenho
          (1) Indicador chave de desempenho
        (g) Coleção de Níveis

(1) Nível
(I) Coleção de Objetos
(A) Objeto
(h) Coleção de Relatórios
(1) Relatório
(I) Instrução de execução de relatório
(II) Resultado da execução de relatório
(III) Coleção de Limiares
(A) Limiar

Estes recursos estão listados de acordo com uma hierarquia que é tida em conta no processo de criação das relações entre os recursos e no processo de criação dos URLs dos recursos. Na base desta listagem de recursos encontra-se o ponto de entrada para as diferentes versões da API. Trata-se de um recurso que é transversal a todas as versões da API e que é necessário disponibilizar para os clientes, para que estes sejam informados das versões disponíveis e de como aceder a cada uma das versões disponíveis.

4.1.1 RELAÇÕES ENTRE OS RECURSOS

Os recursos identificados anteriormente possuem relações entre si, que se vão traduzir em hiperligações presentes nas representações dos recursos. Na figura 4.1 pode-se visualizar as relações existentes, que permitem a navegação pelos recursos da API sabendo apenas o Uniform Resource Locator (URL) do ponto de entrada da API. Em cada relação está indicado o método HTTP que é usado para transitar de um recurso para outro. Assim, no caso em que um cliente da API tenha na sua posse uma representação de uma coleção de adaptações, pode fazer um pedido (GET) para obter uma adaptação específica. O cliente sabe que pode fazer este pedido porque na representação da coleção de adaptações existem hiperligações com esse fim. Estas relações, e consequentemente as hiperligações presentes nas representações, possuem um nome que descreve, dentro do possível, a ação que é realizada. Para não sobrecarregar a figura 4.1, só são apresentados os nomes das relações nos casos em que não é claro quais as ações que são realizadas.

Importa referir que os recursos "Instrução de execução de relatório" e "Resultado da execução de relatório" têm, cada um, duas relações com recursos distintos devido à possibilidade de se poderem obter estes recursos para relatórios que não existam no sistema. Deste modo, quando um cliente obtém a representação de um relatório existente no sistema, esta possui hiperligações para as representações dos recursos "Instrução de execução de relatório" e "Resultado da execução de relatório". No caso em que o cliente queira obter estes recursos para um relatório definido por si que não esteja no sistema, o modo como são acedidos é diferente.

Outras situações que à partida podem não parecer claras são a criação e remoção de recursos. Para clarificar, imagine-se que um cliente quer criar um relatório. Para esse fim, na representação da coleção de relatórios existe uma hiperligação que aponta para a própria coleção de relatórios. Ao fazer o pedido apropriado (POST) para o alvo da hiperligação, o relatório definido no pedido é criado no sistema. A mesma lógica se aplica à remoção de recursos, com a ressalva que o tipo de pedido é diferente (DELETE) e não se aplica a coleções de recursos. Significa isto que, por exemplo, se o cliente quiser apagar todos os relatórios existentes no sistema, tem que o fazer relatório a relatório.

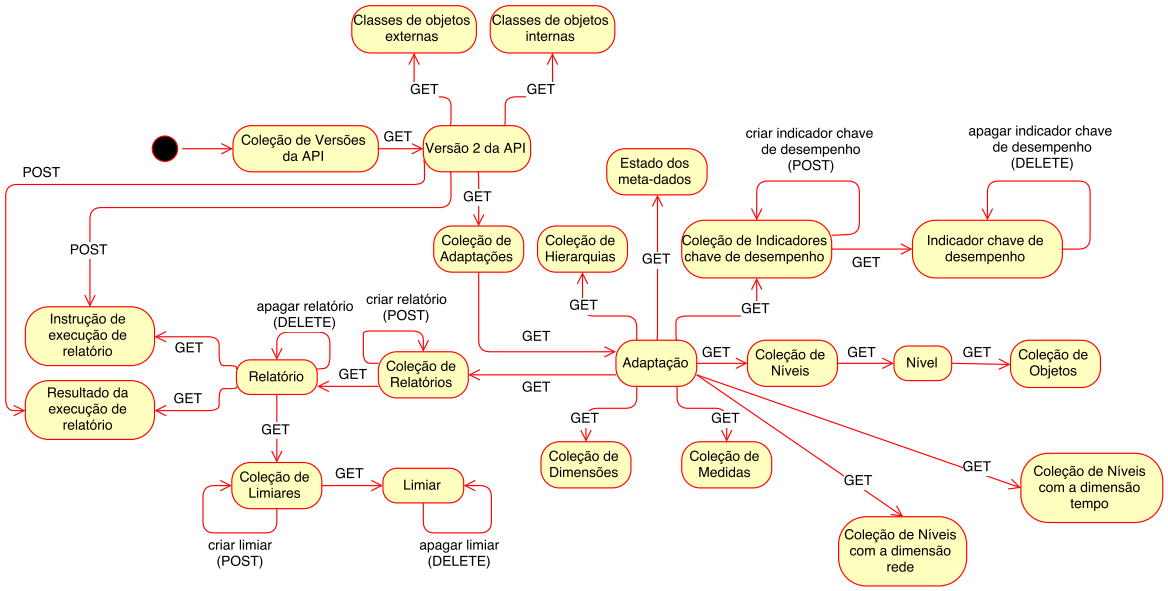

Figura 4.1: Relações entre os recursos da API

4.1.2 REPRESENTAÇÕES DOS RECURSOS

Para representar os recursos, recorreu-se ao HAL+JSON, apresentado no capítulo 2, na secção 2.4.3. Cada representação de um recurso tem um tipo específico de tipo de média que é indicado nos campos *Accept* e *Content-Type* do cabeçalho do protocolo HTTP. Para desenhar o tipo de média para cada representação foi usado o esquema "application/vnd.nokia-[recurso]-v[versão]+json", onde o recurso é o nome do recurso a que o tipo de média se refere e a versão é a versão da representação. Como exemplo, o tipo de média para a primeira versão de uma representação de uma adaptação é "application/vnd.nokia-adaptation-v1+json" Na tabela B.3, do anexo B, pode-se ver o tipo de média para cada recurso da API. Nesta tabela são omitidas as versões de cada tipo de média.

Além do que já foi referido acerca do desenho das representações dos recursos, também foram tidas em conta algumas boas práticas na construção das representações no formato JSON. Parte destas boas práticas são baseadas nas recomendações elaboradas pela Google [34]. Antes de mais, importa referir que o JSON é um formato assente em duas estruturas: coleção de pares chave/valor, comummente denominada por objeto, e lista ordenada de valores. Uma chave é uma string rodeada por aspas e um valor pode ser uma string, um número, um valor booleano (*true* ou *false*), uma lista, um objeto ou o valor nulo (*null*). A listagem 2 mostra a estrutura de um objeto em JSON, composto por um par chave/valor. No caso em que o objeto é composto por mais que um par chave/valor, os pares são separado por uma vírgula. Em relação à lista ordenada de valores, a sua estrutura em JSON pode ser visualizada na listagem 3. Tal como no caso do objeto, se a lista for composta por mais que um valor então os valores são separados por vírgula.

```
{
  "chave": valor
}
```

Listagem 2: Objeto em JSON

```
[
  valor1,
  valor2
]
```

Listagem 3: Lista ordenada de valores em JSON

Dito isto, uma das boas práticas tidas em conta foi a não inclusão de valores na chave de um par chave/valor. A listagem 4 mostra um exemplo em que são usados os números dos colaboradores como chaves. Este tipo de solução traz alguns problemas, nomeadamente a maior dificuldade em ler e interpretar os dados por parte de um cliente da API. Pode-se verificar esta acrescida dificuldade ao comparar a listagem 5, que apresenta a mesma informação mas não tem valores nas chaves, com a listagem 4. Ainda em relação às chaves, estas devem seguir as mesmas convenções que os identificadores em JavaScript. Significa isto que as chaves devem ser strings ascii, em camel case, onde o primeiro caráter deve ser uma letra, um _ ou um $ e os carateres subsequentes podem ser uma letra, um dígito, um _ ou um $. Por fim, devem ser evitadas chaves iguais a palavras reservadas do JavaScript.

```
{
  "colaboradores": [
    {"123": "José"},
    {"456": "Pedro"}
  ]
}
```

Listagem 4: Exemplo de utilização de valores nas chaves em JSON

```
{
  "colaboradores": [
    {"numero": 123, "nome": "José"},
    {"numero": 456, "nome": "Pedro"}
  ]
}
```

Listagem 5: Exemplo de não utilização de valores nas chaves em JSON

4.1.3 URLS DOS RECURSOS

Com o objetivo de disponibilizar os recursos através de URLs intuitivos, foi seguida a estrutura da tabela 4.1 para construir os URLs da API. Além desta estrutura, foram adotadas algumas boas práticas na construção dos URLs que se encontram descritas seguidamente.

Os URLs devem ser compostos por nomes, não devendo incluir verbos. Os verbos denotam ações a executar sobre os recursos e para esse efeito existem os verbos HTTP descritos na tabela 2.2 do capítulo 2.

O nome para uma coleção de recursos deve ser o nome do recurso no plural. Assim, como exemplo, o URL para uma coleção de adaptações deve ser ".../adaptations" e o URL para uma adaptação específica

deve ser ".../adaptations/REPBSS" em que "REPBSS" é o identificador de uma adaptação. Deve-se evitar a utilização de outro léxico para identificar uma coleção, como por exemplo ".../adaptationList".

Os URLs não devem incluir extensões que indiquem o tipo de média do recurso, como por exemplo ".../adaptations/REPBSS.json". Para este efeito devem ser usados os cabeçalhos apropriados do HTTP.

Deste processo de desenho, resultaram os URLs listados na tabela B.1, do anexo B.

URL	**Propósito**
https://{domínio}/api/performanceManager	Versões disponíveis
https://{domínio}/api/performanceManager/v2	Ponto de entrada
https://{domínio}/api/performanceManager/v2/{coleção de recursos}	Coleção de recursos
https://{domínio}/api/performanceManager/v2/{coleção de recursos}/{recurso}	Recurso
https://{domínio}/api/performanceManager/v2/{coleção de recursos}/{?filtro}	Filtra os recursos da coleção
https://{domínio}/api/performanceManager/v2/{coleção de recursos}/{recurso}/{coleção de recursos}	Hierarquia de recursos

Tabela 4.1: Anatomia dos URLs

4.1.4 MÉTODOS HTTP

Na figura 4.1 podem-se visualizar os métodos HTTP usados para obter as representações dos recursos. Para selecionar o método HTTP apropriado a cada caso foi tida em conta a definição dos métodos HTTP que se encontra na especificação do HTTP e que foi descrita no capítulo 2. Assim, para obter representações de recursos, que podem ser coleções de itens ou um item especifico, é usado o método GET. A exceção a esta regra ocorre quando o URL necessário para satisfazer um pedido é demasiado longo. Existe uma limitação no tamanho máximo de um URL, que varia de acordo com o servidor e o *browser* mas que normalmente se considera como sendo 2000 carateres. Nestes casos, à partida, a solução mais óbvia seria incluir a informação no corpo do pedido GET. No entanto, a especificação do HTTP [7] refere que na eventualidade de uma mensagem GET ter corpo, este não tem significado semântico para o pedido. O que isto significa é que caso o servidor altere a sua resposta consoante o que se encontra no corpo do pedido GET, podem existir problemas, por exemplo, ao nível dos *proxies* e do mecanismo de *cache* do HTTP. Por este motivo, a solução de colocar a informação no corpo do pedido GET foi descartada. As restantes soluções consistem em usar o método POST em vez do GET ou em modelar as *queries* como recursos.

A utilização do método POST em vez do GET consiste em fazer um pedido POST em que é colocado dentro do corpo do pedido a informação que seria colocada no URL do pedido GET. Para informar o servidor que este deve tratar o pedido como se fosse um GET, é inserido no pedido POST o cabeçalho X-HTTP-Method-Override com o valor GET, como se pode ver na listagem 6. Quando o servidor recebe o pedido, vai pesquisar quais são os recursos que satisfazem os critérios da pesquisa e envia a resposta tal como aconteceria se recebesse um pedido GET, como se pode visualizar na listagem 7.

```
POST /colaboradores
X-HTTP-Method-Override: GET

{
  "departamento": 5,
  "cargo": "Gestor",
  ...
}
```

Listagem 6: Exemplo de pedido quando se usa o método POST em vez do GET

A modelação das *queries* como recursos tem a vantagem de permitir a reutilização das *queries*. As listagens 8 e 9 exemplificam um pedido para criar uma *query* e a respetiva resposta. Depois da *query* ser criada, o cliente pode obter o resultado da execução da *query* fazendo um pedido GET para "/colaboradores/queries/1/resultado".

Ao comparar as duas soluções é notório que a modelação das *queries* como recursos é mais complexa que a simples utilização do método POST em vez do GET, devido à necessidade de gerir o estado das *queries*. No que respeita à conformidade com o estilo arquitetural REST e com o modelo de maturidade de Richardson, pode-se afirmar que a solução de modelar as *queries* como recursos é melhor que a solução de usar o método POST, isto porque ao usar o método POST em vez do GET não se está a usar devidamente os métodos do HTTP. Para implementar a solução da modelação das *queries* seria necessário gerir o estado das *queries*, o que significa que seria necessário criar novas tabelas na base de dados e desenvolver o código para criar, ler, atualizar e apagar registos dessas tabelas. Apesar de não parecer complicado à partida, num projeto tão complexo como o PM em que parte do código tem bastantes anos, usa tecnologias obsoletas e ainda não está devidamente coberto por testes automáticos,

```
HTTP 200

{
  "_links":{
    "self":{
      "href":"/colaboradores"
    }
  },
  "colaboradores":[
    {
      "nome":"Pedro",
      "numero":456
    },
    {
      "nome":"Ricardo",
      "numero":789
    }
  ]
  ...
}
```

Listagem 7: Exemplo de resposta quando se usa o método POST em vez do GET

```
POST /colaboradores/queries

{
  "departamento": 5,
  "cargo": "Gestor",
  ...
}
```

Listagem 8: Exemplo de pedido quando se modela as *queries* como recursos

```
HTTP 201

{
  "_links":{
    "self":{
      "href":"/colaboradores/queries/1"
    },
    "resultado":{
      "href":"/colaboradores/queries/1/resultado"
    }
  },
  "id":1,
  "departamento":5,
  "cargo":"Gestor"
  ...
}
```

Listagem 9: Exemplo de resposta quando se modela as *queries* como recursos

o esforço de desenvolvimento não compensaria as vantagens da solução. Assim, optou-se pela solução de usar o método POST. É por este motivo, como se pode ver na figura 4.1, que as representações dos recursos "Instrução de execução de relatório" e "Resultado da execução de relatório", para relatórios que não existem no sistema, são obtidas usando o método POST.

Nos restantes casos, os métodos POST são usados para criar recursos. Atualmente, ainda se verifica alguma confusão, no mundo das APIs RESTful, sobre que método, POST ou PUT, se deve usar para criar um recurso. A boa prática que tem emergido e que foi adotada, é que se deve usar o

POST quando é o servidor a decidir qual o identificador que o recurso vai ter, ou seja, quando o cliente, na altura em que faz o pedido para criar o recurso, não sabe qual é o identificador do recurso. Quando o cliente consegue especificar qual o identificador do novo recurso então pode ser usado o método PUT, com a ressalva que tem que ser idempotente. Isto implica que o cliente tenha que enviar sempre a representação completa do recurso que quer que seja criado no servidor pois, caso contrário, o servidor poderia criar recursos diferentes para pedidos iguais, violando a propriedade da idempotência. Como ao criar um relatório e um limiar, o servidor é que tem a responsabilidade de atribuir o identificador ao recurso criado, é usado o método POST.

Finalmente, para apagar relatórios e limiares é usado o método DELETE. Não foi necessário recorrer ao método PATCH pois não existe, neste momento, nenhum requisito que coloque a possibilidade de atualizar um recurso.

4.2 FORMATO DAS DATAS

De modo a evitar más interpretações das datas e horas por parte dos clientes e da própria API, deve ser usado um modo de representar as datas e horas que não seja ambíguo. Com isto em vista, recorreu-se à norma ISO 8601 [35], com o perfil descrito no RFC 3339 [36] para representar todas as datas e horas que estão presentes nas representações ou URLs dos recursos. Na listagem 10 encontra-se o formato, na sintaxe Augmented Backus-Naur Form (ABNF), que está definido no RFC 3339 [36] e que foi adotado para representar as datas e horas.

```
date-fullyear   = 4DIGIT
date-month      = 2DIGIT ; 01-12
date-mday       = 2DIGIT ; 01-28, 01-29, 01-30, 01-31 based on month/year
time-hour       = 2DIGIT ; 00-23
time-minute     = 2DIGIT ; 00-59
time-second     = 2DIGIT ; 00-58, 00-59, 00-60 based on leap second rules
time-secfrac    = "." 1*DIGIT
time-numoffset  = ("+" / "-") time-hour ":" time-minute
time-offset     = "Z" / time-numoffset
partial-time    = time-hour ":" time-minute ":" time-second [time-secfrac]
full-date       = date-fullyear "-" date-month "-" date-mday
full-time       = partial-time time-offset
date-time       = full-date "T" full-time
```

Listagem 10: Formato da data e hora usado na API

Como exemplo, o meio-dia do dia 3, do mês de Setembro, do ano 2015, no fuso horário Coordinated Universal Time (UTC) é representado por “2015-09-03T12:00:00Z”.

4.3 CONDIÇÕES DE ERRO

Sempre que existe uma condição de erro, a API deve enviar uma resposta com o código HTTP adequado ao erro, em que no corpo da resposta está detalhado o erro. Tal como os outros recursos da API, os erros são representados em JSON. Mais especificamente, os erros têm o tipo de média “application/vnd.nokia-error-response+json”. A listagem 11 mostra um exemplo de uma representação

Código interno	Código HTTP	Recursos embutidos
1	401 Unauthorized	-
3	403 Forbidden	-
4	404 Not Found	-
9	409 Conflict	Recurso com o qual existe um conflito

Tabela 4.2: Correspondência entre alguns códigos HTTP e códigos internos

de um erro. A representação do erro é composta por vários campos. O campo "developerMessage" é uma mensagem dirigida aos programadores das aplicações clientes a explicar o erro. O "userMessage" é uma mensagem adequada aos utilizadores finais que pode ser mostrada na interface gráfica de uma aplicação que use a API. O "errorCode" designa o código interno do erro. Finalmente, o "_embedded" possui outros dados que podem ser usados pelo cliente que fez o pedido para melhor analisar o erro e recuperar do erro.

```
{
  "error": {
    "developerMessage": "Message describing the error",
    "userMessage": "End-user message describing the error",
    "errorCode": 190,
    "moreInfo": "http://www.example.com/path/to/help/for/190",
    "_embedded": {
      "embeddedResource": {
      }
    }
  }
}
```

Listagem 11: Representação de um erro

No capítulo 2, na tabela 2.5, já foram referidos alguns códigos de erro HTTP usuais. Os códigos HTTP e os códigos internos que são usados no campo "errorCode" estão alinhados o quanto possível mas não existe necessariamente uma relação de um para um entre cada código HTTP e cada código interno, podendo dar-se o caso de existirem vários códigos internos para o mesmo código HTTP. A tabela 4.2 específica, para alguns casos gerais, quais os códigos internos e o valor do campo "_embedded".

4.4 ESTRATÉGIA DE VERSÕES DA API

Por mais que se tente criar uma API que tenha um tempo de vida prolongado, sem a necessidade de modificações, mais tarde ou mais cedo vão surgir correções ou novos requisitos que levam a que a API tenha que sofrer alterações. Eventualmente, alguma destas alterações irá forçar os clientes a atualizar o seu código ou a alterarem configurações, como é o caso da nova API, tema deste trabalho, em relação à API existente atualmente. Por este motivo, deve ser usado uma estratégia de versões na API que possibilite a um cliente fazer as alterações necessárias para operar com uma nova versão da API antes do tempo de vida da versão anterior terminar.

Existem vários tipos de alterações possíveis de acontecer que justificam diferentes abordagens à estratégia de versões da API. Um tipo de alteração que é bastante provável de acontecer no tempo de vida de uma API é a modificação das representações dos recursos. No caso específico do JSON, caso a alteração da representação consista na adição de um novo par chave/valor a uma representação e os clientes estejam preparados para ignorar pares chave/valor que não reconheçam, então os clientes conseguem continuar a usar a API como até aí a usavam, ignorando os novos pares chave/valor. No entanto, pode-se dar o caso de os clientes não estarem preparados para ignorar pares chave/valor que não reconheçam e nesse caso a adição de um novo par chave/valor poderá fazer com que um cliente deixe de funcionar corretamente. Este funcionamento incorreto dos clientes é bastante provável quando é removido algum par chave/valor de uma representação ou quando a chave de um par chave/valor existente seja modificada ou quando o tipo de valor de um par chave/valor existente seja modificado. Face a este tipo de alterações nas representações, o mais adequado é ter versões para as representações dos recursos. Para indicar a versão de uma representação de um recurso é usado o tipo de média. Assim, um cliente, para obter uma versão específica de uma representação tem que fazer um pedido em que seja indicada a versão no cabeçalho *Accept*. Caso um cliente deseje a ultima versão da representação de um recurso, como atalho, pode indicar o tipo de média "application/json" no cabeçalho *Accept* do pedido. Isto é útil quando se navega pela API com um *Web browser*. Na secção 4.1.2 é explicado a anatomia de um tipo de média e pode-se ver que é incluída uma versão em cada tipo de média.

Numa API RESTful, a modificação dos URLs é algo que não deveria provocar a necessidade de alterações do lado dos clientes. Caso os clientes usem adequadamente as hiperligações presentes nas representações dos recursos então uma alteração de um URL não afeta o cliente. Contudo, é expetável que os clientes decidam atalhar caminho e colocar diretamente no seu código os URLs para todos os recursos da API. Adicionalmente, nas situações em que uma API não implementa devidamente o estilo arquitetural REST, como é o caso da API existente atualmente, a modificação dos URLs, como é imposta pela nova API, vai provocar o funcionamento indevido dos clientes. Devido, principalmente, à razão anteriormente referida, é necessário adotar uma estratégia de versões que permita aos clientes continuar a funcionar corretamente após alterações ao nível do URL. Para estes casos a utilização de versões nas representações não é suficiente e por esse motivo decidiu-se usar versões também ao nível do URL. Deste modo, um cliente que deseje obter, da nova versão da API, as adaptações instaladas, tem que fazer um pedido para o URL ".../v2/adaptations". As versões suportadas podem ser consultadas fazendo um pedido para a ponto de entrada da API, tal como é demonstrado na secção A.1 do anexo A.

Na ótica da estratégia de versões, a API atual e a nova API não são duas APIs distintas mas sim duas versões distintas da mesma API. Em relação à numeração das versões, considerou-se que a API atual é a primeira versão e a nova API é a segunda versão. Uma versão, tanto ao nível da representação do recurso como ao nível do URL do recurso, deve ser um número inteiro precedido da letra "v", como por exemplo v1 ou v2. A versão de uma representação deve ser incrementada quando a representação sofre alguma alteração. A versão presente no URL deve ser incrementada quando existe alguma alteração na API, que não seja alteração de representações, que implique modificações do lado dos clientes.

Como foi explicado na secção 3.6.1 do capítulo 3, a nova versão da API tem que coexistir com a versão existente atualmente durante duas versões do produto PM. A figura 4.2 ilustra o tempo de vida da versão 1 da API com a entrada da versão 2 no produto. Com o fim do tempo de vida da versão 1 da API, esta deixará de estar listada nas versões suportadas que podem ser consultadas a partir do ponto de entrada da API e todos os pedidos feitos para a versão 1 da API resultarão num erro

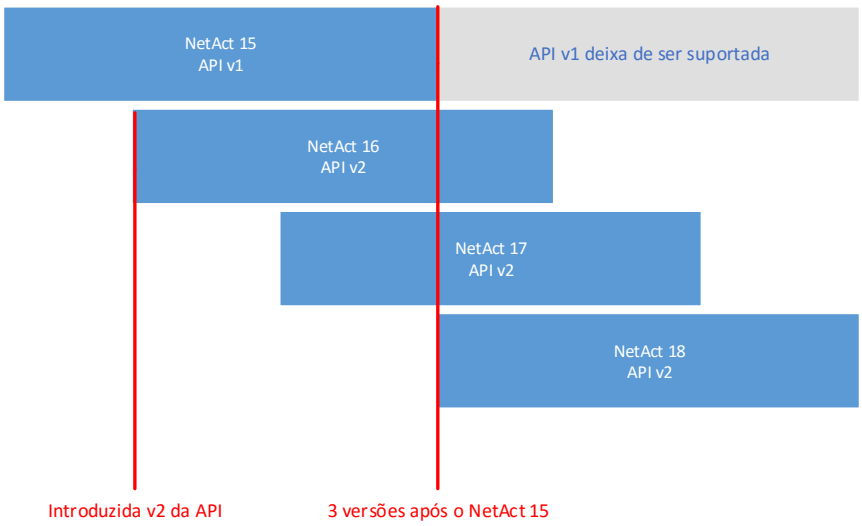

Figura 4.2: Estratégia de versões da API

elucidativo do ocorrido.

4.5 PAGINAÇÃO

A API possui vários métodos que retornam uma coleção de itens para um pedido. Estas coleções podem ser bastante extensas, requerendo uma grande capacidade, por parte dos clientes, para processar as mensagens que contêm as coleções. Adicionalmente, o envio de uma mensagem HTTP extensa com dados que eventualmente nunca serão usados é um desperdício da largura de banda de uma rede.

Para evitar os problemas que advém de enviar coleções completas para os clientes, a solução mais apropriada é dividir uma coleção em várias partes e em cada mensagem enviar apenas uma parte da coleção. A esta solução dá-se o nome de paginação. Nas aplicações *Web*, normalmente, existem duas abordagens distintas para implementar a paginação. A primeira abordagem, que é a abordagem que foi descrita anteriormente, implementa a paginação do lado do servidor. Esta abordagem é indicada quando a quantidade de dados a paginar é bastante elevada, ou se prevê que venha a ser bastante elevada, e tem a vantagem de permitir que os clientes, que podem ser aplicações móveis ou páginas *Web*, apresentem mais rapidamente, aos utilizadores, o primeiro conjunto de itens da coleção. A segunda abordagem consiste no envio das coleções completas para os clientes e estes é que fazem a paginação, apresentando, aos utilizadores, um conjunto de itens de cada vez. Esta abordagem é mais indicada para quando a quantidade de dados a paginar é reduzida e tem a vantagem de permitir aos clientes apresentarem mais rapidamente os conjuntos de itens seguintes ao primeiro conjunto. Devido à elevada quantidade de dados que certas coleções podem conter, para o caso da nova API do PM, optou-se por implementar a paginação do lado do servidor. Deste modo, a API, por omissão, retorna até 30 itens por cada pedido feito por um cliente a um dos métodos que retorna uma coleção de itens. Caso existam mais itens, a API fornece ligações para navegar para os restantes itens da coleção.

Usualmente, a paginação é implementada usando um esquema baseado em páginas. A figura 4.3

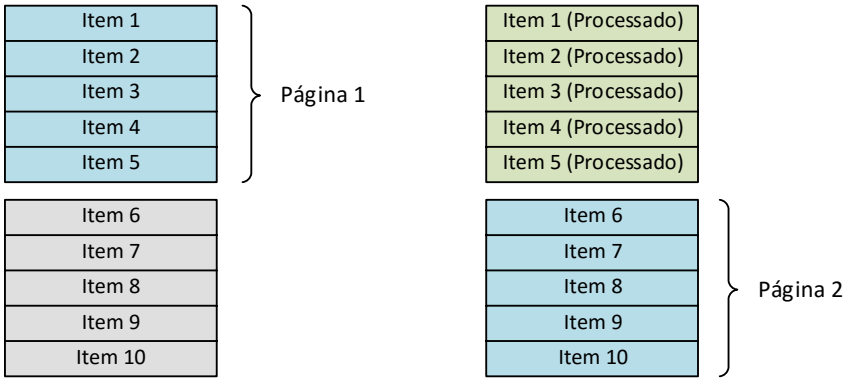

Figura 4.3: Paginação usando um esquema baseado em páginas

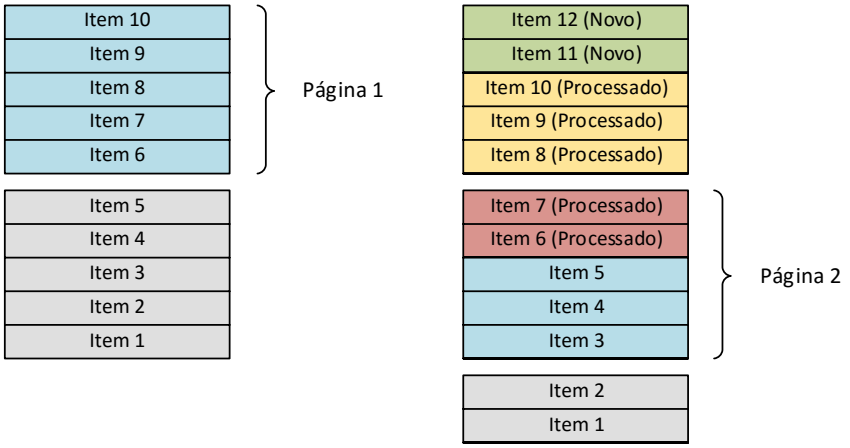

Figura 4.4: Problema da paginação usando um esquema baseado em páginas

demonstra o funcionamento do esquema baseado em páginas. Neste esquema, a coleção é partida em várias páginas tendo em conta apenas o tamanho da coleção e o número de itens a retornar de cada vez, ignorando se um item já foi retornado ou não. Por este motivo, quando a composição de uma coleção é alterada frequentemente, ora através da adição de novos dados ou a remoção de dados existentes, este tipo de esquema pode levar a que o servidor envie itens repetidos em diferentes páginas da coleção, como é ilustrado na figura 4.4. No exemplo da figura 4.4, a API retorna os itens ordenados do mais recente para o mais antigo e entre o primeiro e o segundo pedido do cliente surgem dois novos itens. Como só é tido em conta o número de itens em cada página, vão existir dois itens que se repetem na segunda resposta da API.

Devido ao problema referido anteriormente e como em alguns casos os dados do PM podem ser alterados frequentemente, descartou-se o esquema baseado em páginas e optou-se por usar um esquema de paginação baseado em cursores. Neste esquema, os itens são retornados de acordo com a posição de um cursor na coleção. Assim, quando um cliente faz um pedido ao servidor, este envia o primeiro conjunto de itens e a posição em que o cursor se encontra, que será o item seguinte ao primeiro conjunto. No pedido seguinte, o cliente pode usar o cursor para indicar que quer o segundo conjunto de itens

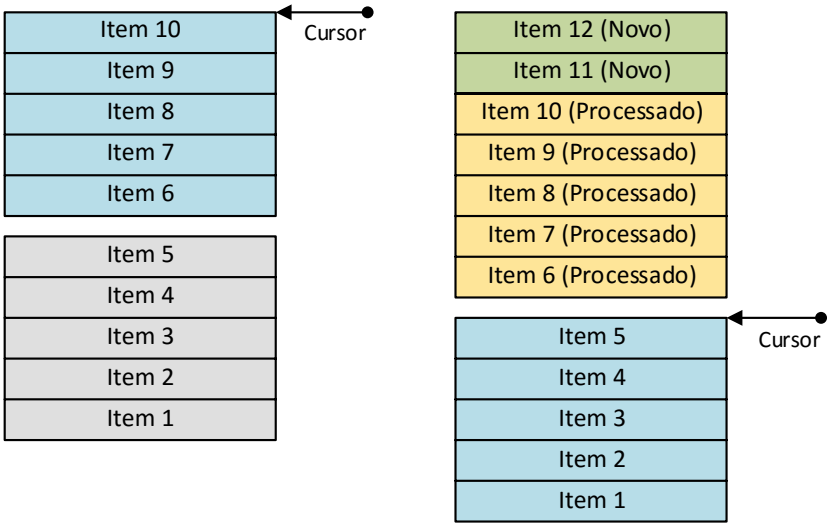

Figura 4.5: Paginação usando um esquema baseado em cursores

a começar na posição do cursor. A figura 4.5 demonstra o funcionamento do esquema baseado em cursores e a maneira como evita os problemas apresentados pelo esquema baseado em páginas.

Um cliente pode navegar por uma coleção recorrendo a dois parâmetros de *query* do URL que são o *before* e o *after*. Ao usar um destes parâmetros, o cliente tem que fornecer um identificador de um cursor como valor do parâmetro, como se pode ver na listagem 12. Assim, quando um cliente coloca no URL de um pedido o parâmetro *after* e um identificador de um cursor, a API retorna os itens subsequentes à posição do cursor. Este modo de funcionamento é igual para o parâmetro *before*, com a diferença que a API retorna os itens anteriores à posição do cursor. Os clientes não devem fazer nenhuma suposição sobre os valores que os identificadores de cursores podem ter e sobre a sua duração, podendo dar-se o caso de um identificador ser gerado aleatoriamente pela API ou ficar inválido após um determinado período de tempo.

Em cada representação de um coleção, retornada pela API, existem hiperligações que o cliente pode usar para navegar pela coleção. Caso o cliente deseje os seguintes itens de uma coleção então pode aceder à relação *next* e usar a hiperligação associada e caso deseje os itens anteriores pode recorrer à relação *previous*. A inclusão destas hiperligações depende da posição atual do cursor, ou seja, caso o cliente aceda aos primeiros itens de uma coleção, essa representação não irá conter a hiperligação com a relação *previous*. O mesmo é verdade para a representação dos últimos itens da coleção e a hiperligação com a relação *next*. Este comportamento pode ser visualizado na figura 4.6. Ainda na listagem 12 pode-se verificar que existe mais um parâmetro de *query* nos URLs das hiperligações, de seu nome *limit*. Este parâmetro indica à API qual o número de itens a retornar em cada conjunto de uma coleção e como já foi referido anteriormente, a API, por omissão, atribui o valor trinta a este parâmetro. Um exemplo de uma interação entre o cliente e a API, que demonstra como é que a paginação deve funcionar, pode ser visualizado nas listagens 13, 14, 15 e 16.

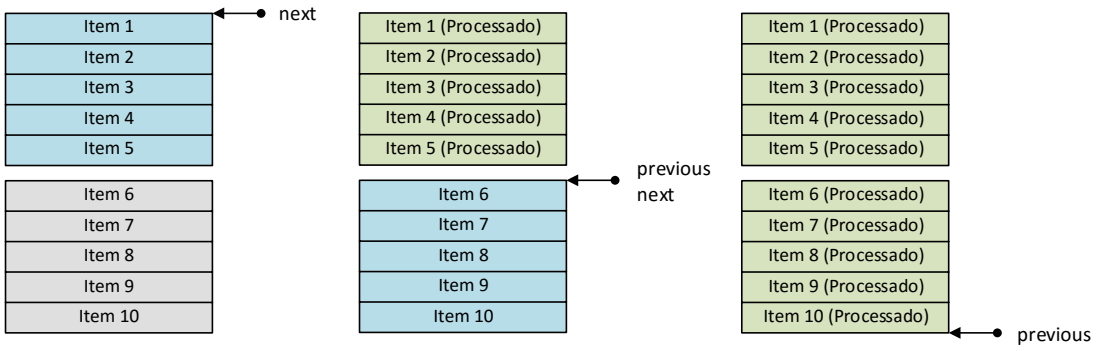

Figura 4.6: Relações das hiperligações usadas na paginação

```
{
  "_links": {
    "next": { "href": "/users?after=MTAxNTExOTQ1MjAwNzI5NDE&limit=30" },
    "previous": { "href": "/users?before=MTAxNTExOTQ1MjAwNzI5NDE&limit=30" }
  }
}
```

Listagem 12: Representação de uma coleção com paginação

```
GET /adaptations
Accept: application/vnd.nokia-adaptations+json
```

Listagem 13: Pedido para obter uma coleção

```
HTTP 200
Content-Type: application/vnd.nokia-adaptations+json

{
  "_links":{
    "next":{
      "href":"/adaptations?after=VTUzCTEwQOT1jMwPLxY2RJH&limit=30"
    }
    ...
  },
  "_embedded":{
    "nokia:adaptations":[
      {
        "id":"nokacs_content"
        ...
      },
      {
        "id":"RASWPM"
        ...
      }
    ]
  }
  ...
}
```

Listagem 14: Resposta com os primeiros itens de uma coleção

```
GET /adaptations?after=VTUzCTEwQOT1jMwPLxY2RJH&limit=30
Accept: application/vnd.nokia-adaptations+json
```

Listagem 15: Pedido para obter os seguintes itens uma coleção

```
HTTP 200
Content-Type: application/vnd.nokia-adaptations+json

{
  "_links":{
    "previous":{
      "href":"/adaptations?before=ZXCwASDzUIP9fNwRTxG5JKL&limit=30"
    }
    ...
  },
  "_embedded":{
    "nokia:adaptations":[
      {
        "id":"NOKAXC"
        ...
      },
      {
        "id":"NOKLTE"
        ...
      }
    ]
  }
  ...
}
```

Listagem 16: Resposta com os seguintes e últimos itens de uma coleção

4.6 SEGURANÇA DA API

Mais frequentemente do que é desejável, a segurança de um sistema de informação é relegada para segundo plano, sendo ignorada durante o desenho do sistema. Apenas quando o sistema está em vias de entrar em produção ou ser integrado com outros sistemas é que a segurança é tida em conta. Esta abordagem pode abrir a porta a falhas graves de segurança, ainda mais graves quando o sistema é crítico, como é o caso dos sistemas de suporte à operação. Por este motivo, a segurança deve ser uma parte integrante do desenho da API.

No que concerne à autenticação dos clientes da API, existem várias soluções que podem ser usadas. A solução mais fácil de implementar, mas também a menos segura, é a utilização do HTTP Basic Access Authentication. Neste esquema, o user-ID e a palavra-passe de cada cliente é enviada, em texto normal, na mensagem HTTP, à vista de quem intercetar a mensagem. Por este motivo, este esquema só apresenta alguma segurança quando é usado sobre um canal de comunicação encriptado, como é o caso de quando é usado Transport Layer Security (TLS). Para ultrapassar este problema, foi criado o esquema Digest Access Authentication. Neste esquema, um valor calculado a partir da palavra-passe do cliente é enviado na mensagem HTTP em vez da própria palavra-passe e por este motivo, o Digest não depende de nenhum esquema de segurança ao nível do transporte para ser considerado seguro. Outros tipos de soluções foram surgindo ao longo dos anos, como é o caso do Security Assertion Markup Language (SAML) [37] ou do OpenID [38]. Estas soluções são mais seguras porque requerem as credencias do cliente apenas numa fase inicial sendo que posteriormente é usado um *token* para identificar o cliente. Ultimamente, o OpenID Connect tem ganho alguma atração em companhias como a Google e a Microsoft, que o passaram a usar. O OpenID Connect consiste numa simples camada de autenticação no topo do protocolo OAuth 2.0. Para a API ainda não está definido qual o esquema de autenticação a usar, mas face ao estado da arte dos esquemas de autenticação, seria aconselhável usar o OpenID Connect.

Em relação à autorização dos clientes, sem sombras de dúvida que o OAuth 2.0 é o protocolo mais usado nas APIs públicas devido à sua segurança e facilidade de uso [39]. Por esses motivos, o OAuth 2.0 deve ser o protocolo de autorização a usar na nova versão da API do PM. Atualmente, a plataforma sobre a qual assenta o PM não está preparada para suportar o OAuth 2.0. No entanto, vai-se proceder, a curto prazo, à migração da plataforma para uma que já suporta o OAuth 2.0.

Outro ponto fulcral para implementar uma API com o nível de segurança requerido pela Nokia é assegurar a privacidade e integridade dos dados que são transmitidos entre a API e os clientes. Com esses objetivos em mente, optou-se por usar, nas comunicações entre a API e os clientes, HTTP sobre TLS, ou, como também é conhecido devido ao identificador de protocolo usado no URI, HTTPS [40], [41]. No TLS, a privacidade de uma conexão é assegurada através da encriptação dos dados que são transmitidos na conexão e a integridade dos dados é alcançada com a inclusão, nas mensagens transmitidas, de um valor de verificação de integridade da mensagem. Além das vantagens ao nível da segurança, na escolha do TLS também pesou o facto de algumas implementações do HTTP 2, tais como os *browsers Web*, só planearem suportar o HTTP 2 quando este é usado sobre uma conexão encriptada [42], [43].

Além dos tópicos já abordados, existem outras boas práticas que podem ser seguidas para implementar uma API RESTful segura [44]. Por exemplo, a API deve validar as representações enviadas pelos clientes de modo a evitar possíveis ataques em que é inserido algum valor na representação que provoca um comportamento indesejado do servidor. Outro exemplo é a inclusão do cabeçalho "X-Content-Type-Options: nosniff" nas respostas enviadas pela API, evitando-se assim alguns ataques relacionados com os tipos de média. Outra prática que se deve usar para prevenir que os clientes

obtenham mais informações do que é suposto é evitar usar os erros com o *status code* 401 e 403 do HTTP para pedidos não autorizados porque o uso destes erros informa, em primeira instância, que o recurso pedido existe, o que se qualifica como uma fuga indesejada de informação. Para estes casos, o erro retornado pela API deve ser o 404 [7], [45].

Um ponto importante para se entender a causa de alguns problemas, entre os quais falhas de segurança, que podem afetar um sistema de informação é a utilização de registos (*logs*). De cada vez que um utilizador realiza alguma ação relevante no sistema, este deve registar várias informações, entre elas quem é que realizou a ação, qual a ação realizada e a data em que a ação foi realizada. Estes registos, além de serem úteis quando se está a tentar descobrir a causa de um erro no sistema, tendo em vista a sua correção, também são importantes para detetar acessos não autorizados no sistema. Por este motivo, é desejável que a API implemente um sistema de registos que, numa primeira fase, seja fácil de compreender. Numa segunda fase, os registos terão que estar de acordo com alguns requisitos, nesta área, que estão neste momento a ser definidos pela Nokia para todos os componentes do NetAct. Esses requisitos envolvem, entre outras coisas, que sempre que ocorra um evento, o registo do evento seja enviado para um componente do NetAct responsável por recolher os registos dos eventos.

4.7 ARQUITETURA

Tendo em consideração os atributos de qualidade e os casos de uso descritos no capítulo 3, foi elaborada uma arquitetura para a API. Esta arquitetura é explicada seguidamente com recurso a diferentes vistas, cada uma realçando diferentes aspetos que contribuem para os atributos de qualidade [46], [47].

Na figura 4.7 é apresentado os diferentes módulos que constituem a arquitetura da API. O módulo "Controller" é o ponto de contacto com os clientes, recebendo e mapeando os pedidos dos clientes e enviando as devidas respostas. Para poder satisfazer os pedidos dos clientes, o módulo "Controller" comunica com o módulo "Gateway" e com o módulo "Model". O módulo "Gateway" comunica com os serviços do PM e com o módulo "Model". No módulo "Model", mais especificamente no módulo "PM Model" encontram-se as entidades retornadas pelo PM. Estas entidades são traduzidas para entidades ajustados à API. Este processo é feito no módulo "Model Assembler" e o resultado da conversão encontra-se no módulo "API Model". Por sua vez, estas entidades resultantes da conversão são convertidas em recursos com controlos hipermédia que se encontram no módulo "Resource". Esta conversão é feita no módulo "Resource Assembler". Por cada conceito do domínio, como por exemplo uma adaptação, existe, em cada módulo da figura 4.7, uma classe específica. Quer isto dizer que, apenas para citar dois exemplos, no módulo "Controller" existe uma classe "AdaptationController" que recebe os pedidos referentes às adaptações e existe uma classe "DimensionController" que recebe os pedidos referentes às dimensões. Esta lógica aplica-se a todos os módulos apresentados na figura 4.7. A figura 4.8 demonstra esta lógica, apenas para o caso das adaptações. As classes para os outros conceitos foram omitidas.

Para se perceber toda a sequência de operações que são efetuadas, nos módulos e classes anteriormente referidos e que são mostrados na figura 4.8, desde que um pedido é feito à API até que esta envie a resposta, é mostrado na figura 4.9 um diagrama de sequência para o caso em que um cliente faz um pedido para obter todas as adaptações. Quando um pedido para obter todas as adaptações chega ao servidor, o pedido é mapeado num método da classe "AdaptationController".

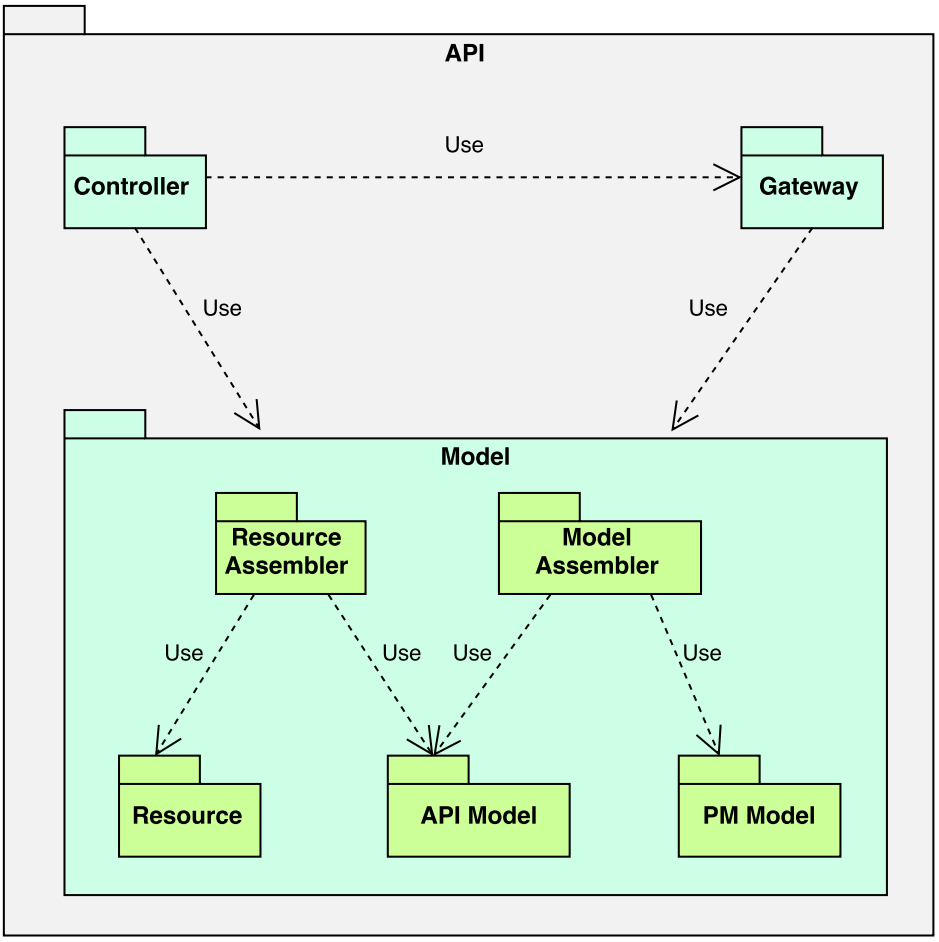

Figura 4.7: Módulos da API

Esse método, por sua vez, vai invocar um método da classe "AdaptationGateway" que faz a ponte entre a API e os serviços do PM. A classe "AdaptationGateway" invoca o serviço devido do PM, neste caso o serviço "KPIExportService", que retorna uma lista de instâncias da classe "TechnologyVO". A "TechnologyVO" é uma classe bastante diferente do que deve ser servido aos clientes da API quando estes pedem uma adaptação. Por este motivo, após a classe "AdaptationGateway" receber a lista de "TechnologyVO" como retorno da chamada ao método da classe "KPIExportService", a classe "AdaptationGateway" invoca um método da classe "AdaptationAssembler" para converter a lista de instâncias da classe "TechnologyVO" numa lista de instâncias da classe "Adaptation". Após esta conversão, a classe "AdaptationGateway" retorna esta lista de instâncias da classe "Adaptation" para o "AdaptationController". Apesar da classe "Adaptation" já estar num formato adequado para os clientes da API perceberem, ainda não possui os controlos hipermédia necessários, tais como as hiperligações relacionadas com a paginação. Por essa razão, a classe "AdaptationController" invoca um método da classe "AdaptationResourceAssembler" para converter a lista de instâncias da classe "Adaptation" para uma lista de instâncias da classe "AdaptationResource", classe esta que já contém os controlos hipermédia. Finalmente, o "AdaptationController" usa a lista de instâncias da classe "AdaptationResource" para construir a representação com o tipo de média (JSON) negociado com o cliente que fez o pedido.

A API para efetuar as operações requisitadas pelos clientes necessita de comunicar com os serviços

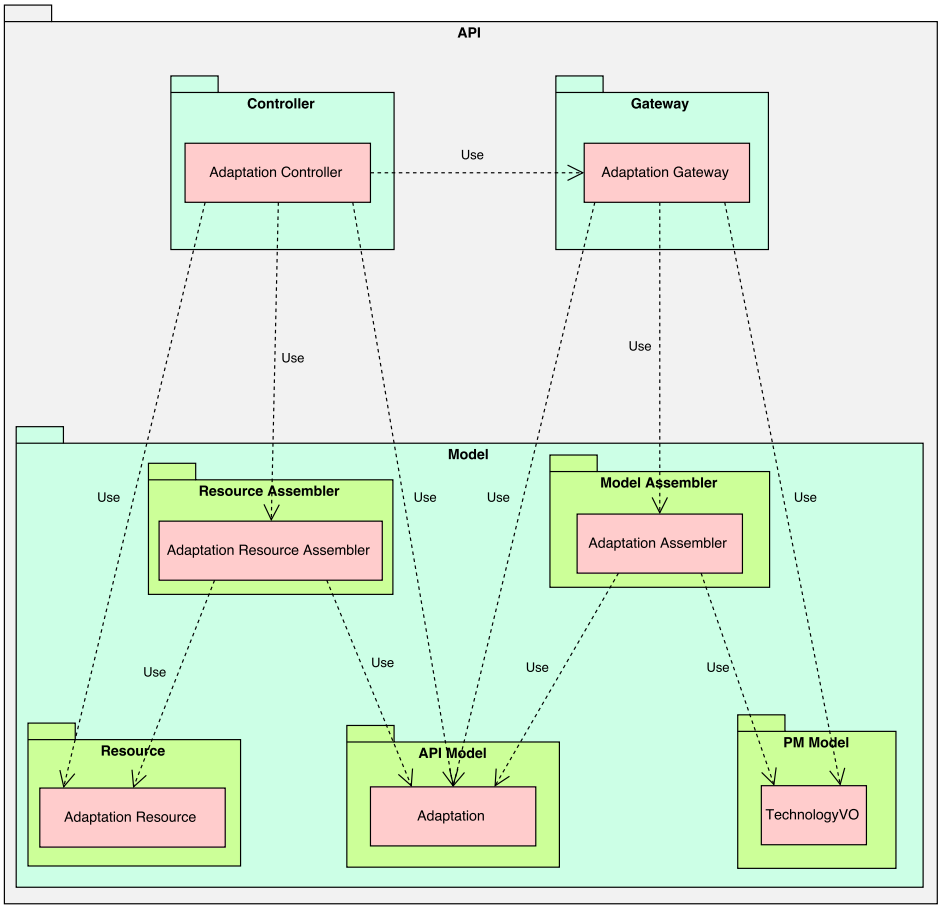

Figura 4.8: Decomposição dos Módulos da API

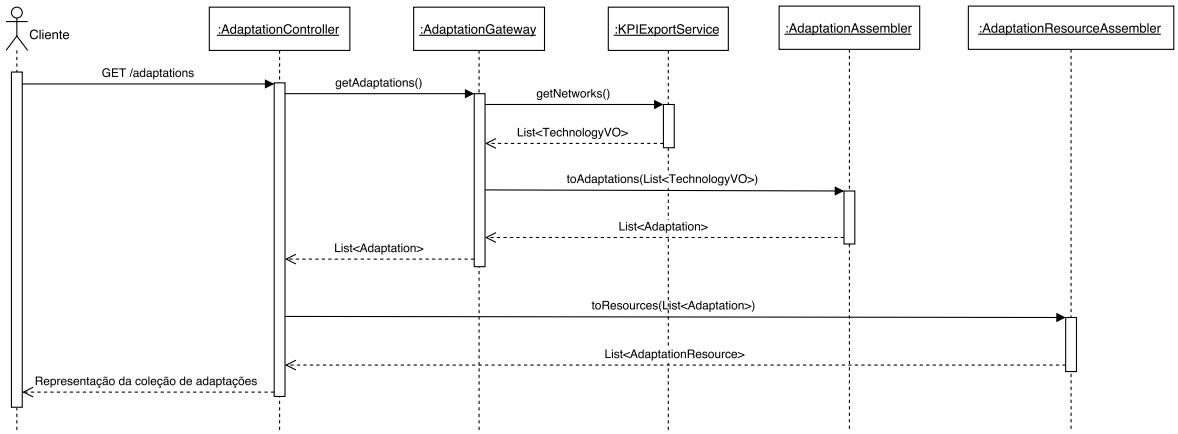

Figura 4.9: Diagrama de Sequência - Obter todas as adaptações

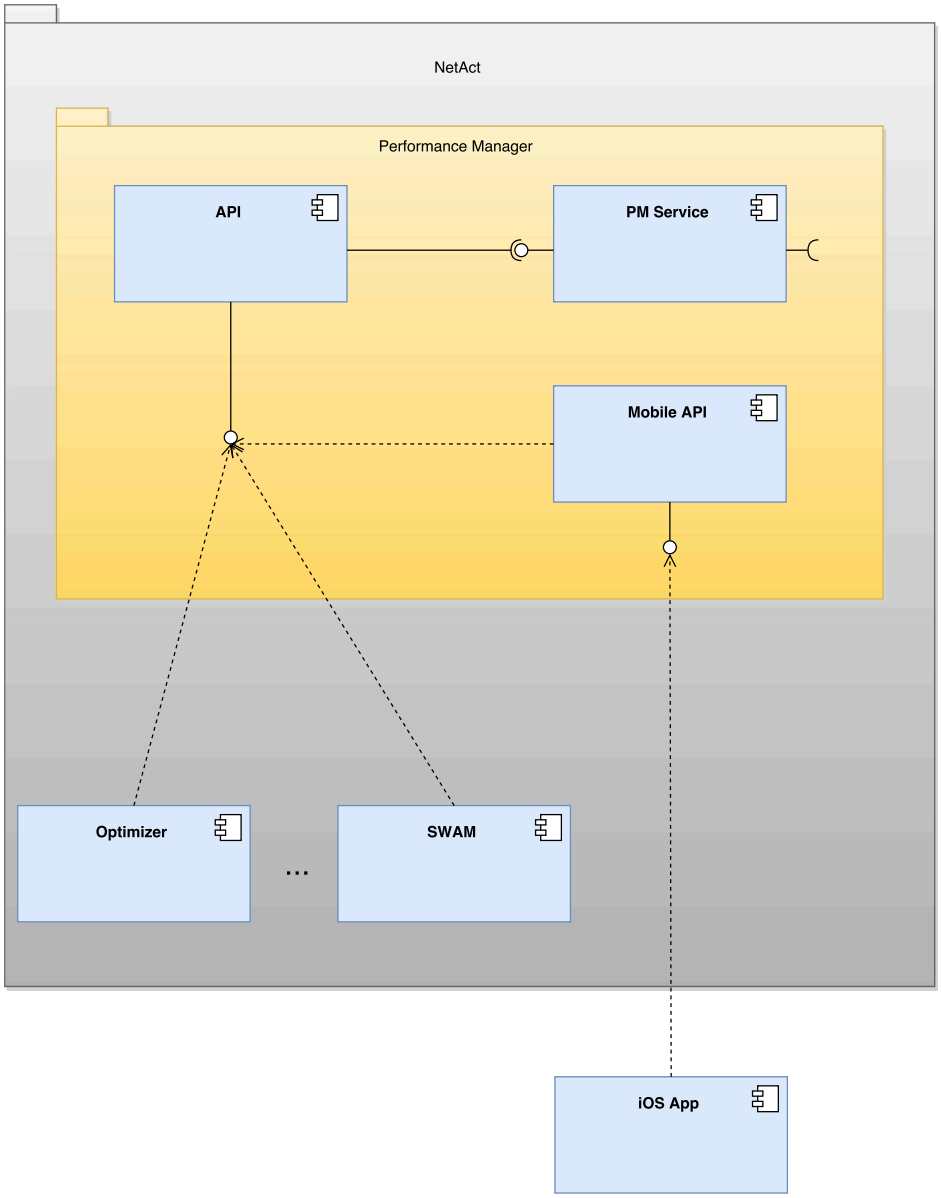

Figura 4.10: Componentes e conetores da API

do PM, tal como mostra a figura 4.10. São estes serviços do PM, que por sua vez invocam outros componentes, que permitem obter os dados guardados na base de dados do PM e operar sobre eles. O "KPIExportService" referido anteriormente e que se pode observar na figura 4.9 é um caso de um serviço do PM. A API, por sua vez, fornece uma interface RESTful para os clientes. Estes clientes podem ser desde aplicações do NetAct que necessitam de comunicar com o PM, tais como o Optimizer ou o SWAM, até à *Mobile* API usada pela aplicação iOS em desenvolvimento, passando pelo portal *Web* do PM. A aplicação iOS invoca a *Mobile* API e esta, por sua vez, invoca a API para obter os dados necessários para satisfazer o pedido da aplicação iOS.

Na figura 4.11 pode ser visto como é que a API é instalada no servidor. Convém notar que a API é empacotada dentro do artefacto "performanceManager.ear". O servidor do PM é decomposto em três servidores, sendo que a aplicação PM é instalada no servidor aplicacional, que neste caso é o WebSphere [33].

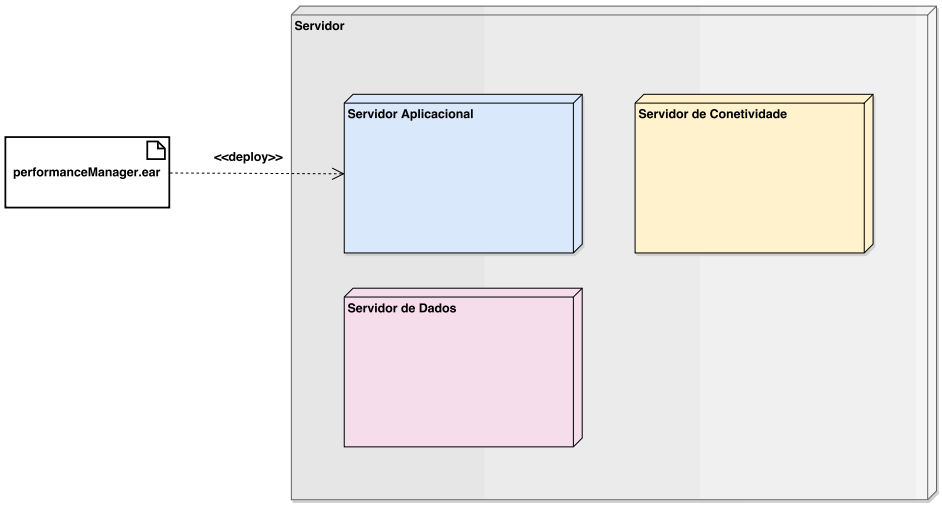

Figura 4.11: Diagrama de instalação da API

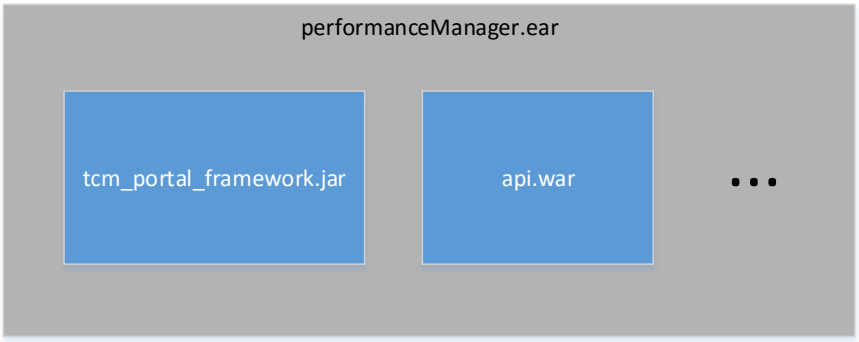

Figura 4.12: Composição do artefacto performanceManager.ear

A figura 4.12 mostra a composição do artefacto "performanceManager.ear" que foi referido anteriormente e que é instalado no servidor aplicacional, como se pode visualizar na figura 4.11. Dentro do artefacto "performanceManager.ear" encontra-se o artefacto "api.war" que possui o código fonte e os ficheiros de configuração da segunda versão da API. Na figura 4.12, para além do artefacto "api.war", só é mostrado o artefacto "tcm_portal_framework.jar" que contém os serviços do PM que a API invoca e também contém a primeira versão da API. Quando a primeira versão da API deixar de ser suportada, o código da primeira versão da API será removido do artefacto "tcm_portal_framework.jar". O artefacto "performanceManager.ear" é composto por bastantes mais artefactos que aqueles que são mostrados na figura 4.12, mas esses artefactos foram omitidos porque não são relevantes para esta discussão da arquitetura.

capítulo 5

Implementação da API

Este capítulo apresenta um subconjunto das classes de *software* implementadas. Posteriormente são descritas as principais metodologias e técnicas de desenvolvimento usadas, bem como as tecnologias adotadas para desenvolver a API.

5.1 classes de software

Nesta secção é descrito, em maior pormenor, as classes que foram apresentadas na secção de arquitetura do capítulo anterior. Para esta descrição só foram tidas em conta as classes relevantes para quando são invocados os métodos da API que retornam uma coleção de adaptações e uma adaptação específica. Para as outras entidades do domínio, como por exemplo o relatório, a estruturação das classes é semelhante ao caso aqui apresentado.

Na figura 5.1 pode-se ver um diagrama de classes, em Unified Modeling Language (UML), focado nos casos de uso para obter adaptações. Adicionalmente, nas tabelas 5.1, 5.2, 5.3, 5.4 e 5.5 são descritos os métodos públicos de algumas das classes representadas na figura 5.1. Os métodos privados, os construtores e os métodos públicos que não são relevantes para esta discussão, tais como o "equals" ou o "toString", foram omitidos do diagrama e das tabelas. A classe "AdaptationResource" não é descrita porque se trata de uma classe com apenas um construtor, que estende uma classe do Spring, onde é encapsulada a classe "Adaptation" e as hiperligações da representação da adaptação. A classe "TechnologyVO" é uma das classes devolvidas pelos serviços internos do PM e por esse motivo não é apresentada em todo o seu detalhe.

O diagrama da figura 5.2 exibe as classes envolvidas no mecanismo de tratamento de erros da API, focado nas adaptações. Existe uma classe, "AdaptationErrorHandler", que apanha todas as exceções lançadas pela API, relacionadas com as adaptações, e constrói a mensagem de erro apropriado para enviar para o cliente. Para cada tipo de erro que a API pode retornar, existe um método associado na classe "AdaptationErrorHandler" que apanha as exceções que estão na origem do erro e constrói uma instância da classe "ApiError" apropriada para o erro em causa. Para todos os outros erros, que não estão relacionados com as adaptações, existem outras classes parecidas com a "AdaptationErrorHandler".

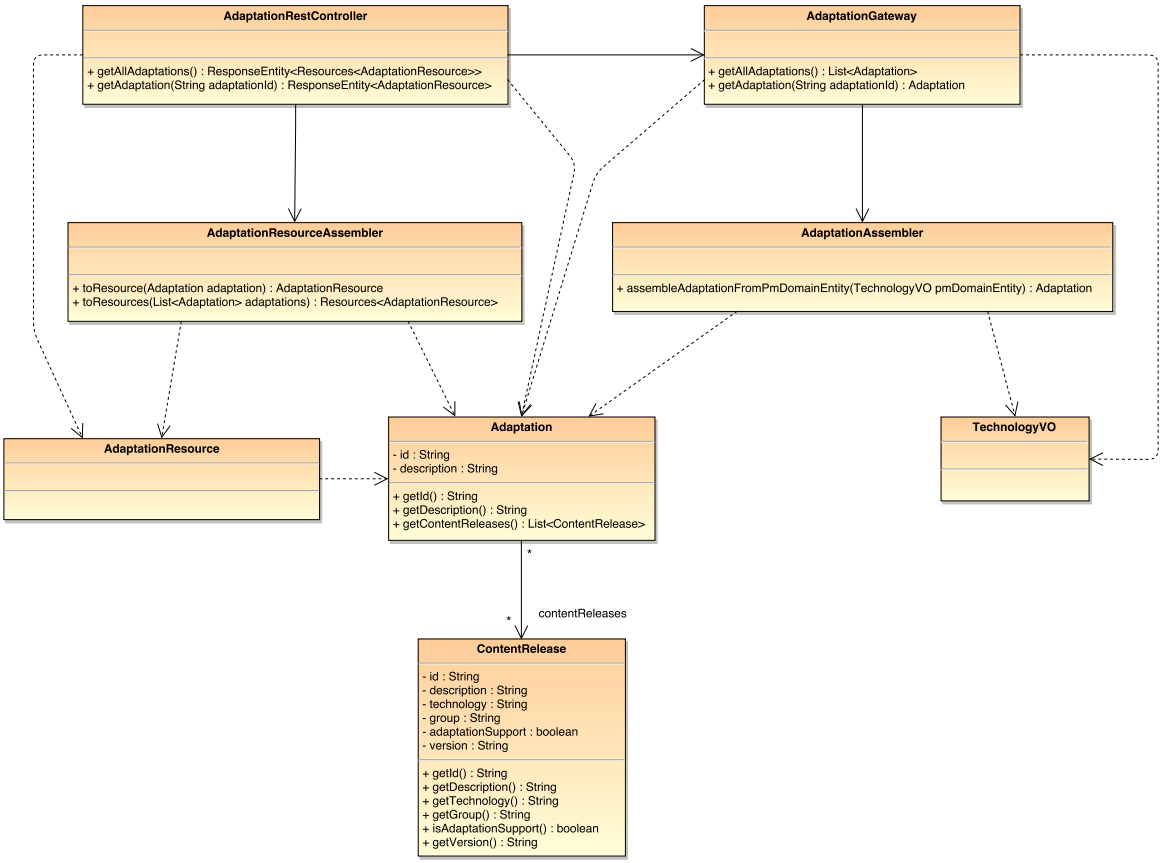

Figura 5.1: Diagrama de classes focado nos casos de uso para obter adaptações

AdaptationRestController	
getAllAdaptations()	Retorna uma representação da coleção de todas as adaptações
getAdaptation(String adaptationId)	Retorna uma representação da adaptação correspondente ao id fornecido

Tabela 5.1: Métodos da classe AdaptationRestController

Adaptation	
getId()	Retorna o id da adaptação
getDescription()	Retorna a descrição da adaptação
getContentReleases()	Retorna todas as versões dos pacotes de conteúdo associados à adaptação

Tabela 5.2: Métodos da classe Adaptation

AdaptationResourceAssembler	
toResource(Adaptation adaptation)	Converte uma adaptação num recurso
toResources(List<Adaptation> adaptations)	Converte uma coleção de adaptações num recurso

Tabela 5.3: Métodos da interface AdaptationResourceAssembler

AdaptationGateway	
getAllAdaptations()	Retorna todas as adaptações disponíveis no PM
getAdaptation(String adaptationId)	Retorna a adaptação correspondente ao id fornecido

Tabela 5.4: Métodos da interface AdaptationGateway

AdaptationAssembler	
assembleAdaptationFromPmDomainEntity(TechnologyVO pmDomainEntity)	Constrói uma adaptação com base na entidade retornada pelos serviços do PM

Tabela 5.5: Métodos da classe AdaptationAssembler

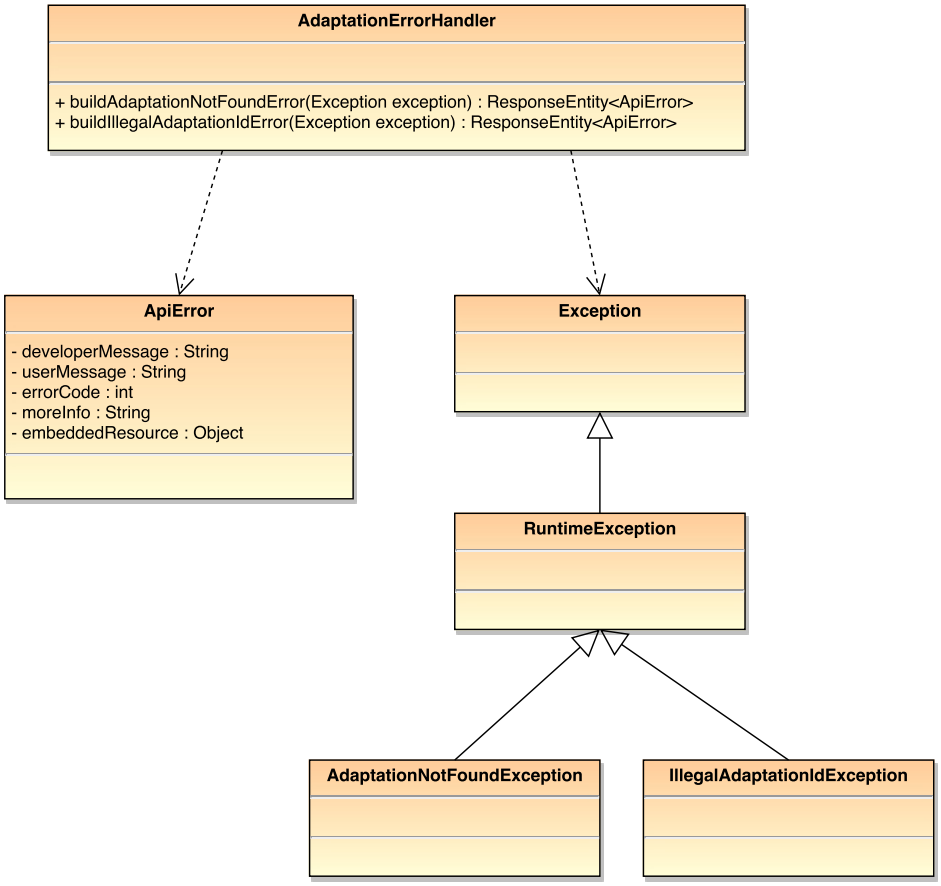

Figura 5.2: Diagrama de classes focado no tratamento dos erros

5.2 TESTES

O código foi produzido seguindo o processo de desenvolvimento Test Driven Development (TDD). Para testar um sistema de informação é possível aplicar diferentes tipos e níveis de teste. No caso da API, para já, apenas foram implementados testes de unidade. Os testes de unidade isolam e verificam uma pequena parte do código do sistema. Normalmente, quando se utiliza o paradigma da programação orientada a objetos, considera-se que a unidade corresponde a uma classe. No futuro, provavelmente, serão criados testes de sistema ou *end-to-end* para testar completamente a integração da API com o resto do sistema.

No diagrama da figura 5.3 é exibido uma classe, de teste de unidade, que testa a classe "AdaptationResourceAssembler". Como se pode ver, a classe é composta por inúmeros métodos para testar "todos" os casos possíveis que podem ocorrer. Na frase anterior, a palavra "todos" encontra-se entre aspas porque só se consegue testar o que se consegue conceber que possa acontecer no sistema. Existem sempre casos que escapam à imaginação de quem cria os testes. Na figura 5.3, são omitidos os modificadores de visibilidade e o tipo de retorno de todos os métodos da classe "AdaptationResourceAssemblerTest" porque são todos públicos e retornam *void*.

Ao escrever os testes, uma das métricas que se teve em mente foi a percentagem do código coberto pelos testes. O objetivo foi ter uma cobertura próxima de 100%. Não se assumiu exatamente os 100% porque existem sempre algumas classes que não faz sentido testar, tais como classes de configuração do Spring. Na figura 5.4 é exibido a cobertura atual da API, recolhida com a ferramenta EclEmma no Eclipse [48]. A cobertura não é uma garantia que os testes estão a testar devidamente o código. Pode-se dar o exemplo de testes em que simplesmente é invocado o método a testar e não é feito nenhum tipo de verificação. Nestes casos, as ferramentas que analisam a cobertura assumem que o método a testar está coberto por testes. Existem técnicas para verificar a qualidade dos testes mas até agora não foi adotada nenhuma. No futuro poderá fazer sentido utilizar a técnica de testes de mutação. Esta técnica consiste em criar versões do código com alterações em alguns pontos e verificar se os testes falham por causa dessas alterações, como é suposto acontecer se os testes forem bem-feitos.

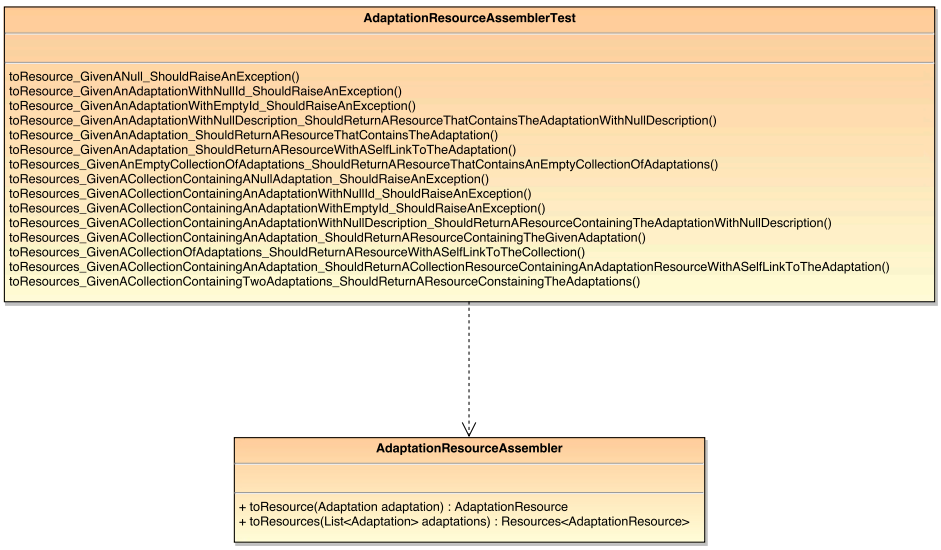

Figura 5.3: Exemplo de classe de teste de unidade

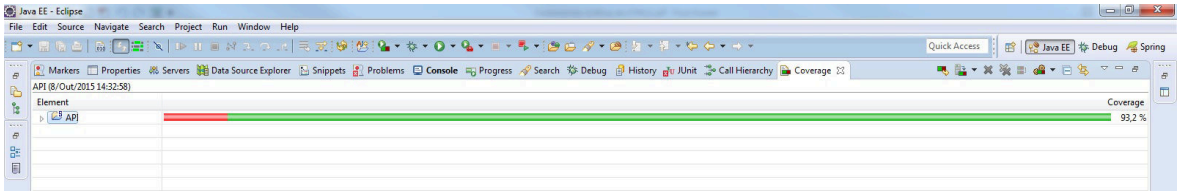

Figura 5.4: Percentagem do código da API coberto por testes

5.3 METODOLOGIAS E TÉCNICAS DE DESENVOLVIMENTO

No desenvolvimento da API foram seguidas algumas metodologias e usadas várias técnicas de desenvolvimento tendo em vista a produção de uma API com os padrões de qualidade exigidos pela Nokia. Seguidamente, são abordadas estas metodologias e técnicas.

5.3.1 SCRUM

Scrum é uma metodologia de desenvolvimento ágil, ou seja, é consistente com os princípios do manifesto ágil [49]. O Scrum é usado para gerir o processo de desenvolvimento de software. Um dos principais conceitos do Scrum é a noção de *sprint*. Uma *sprint* é um período de tempo, usualmente duas semanas, em que é produzido algo com qualidade suficiente para poder ser entregue ao cliente. No caso da equipa da Nokia onde foi inserido o desenvolvimento da API, as *sprints* têm tido uma duração de três semanas e consequentemente também foi adotada uma *sprint* de três semanas para o desenvolvimento da API. No início de cada *sprint* existe uma reunião, denominada de *sprint planning*, para planear o que vai ser feito durante a *sprint* e no final de cada *sprint*, o trabalho feito durante a *sprint* é demonstrado para todos os intervenientes numa atividade denominada de *sprint review*. Também no final da *sprint*, é feita uma reunião, denominada de *sprint retrospective*, onde se discute como é que correu a *sprint* e os pontos a melhorar nas *sprints* futuras. Diariamente, durante a *sprint*, é feita uma reunião de curta duração, denominada de *daily scrum*, entre os elementos da equipa de desenvolvimento, para estes se sincronizarem e alinharem o trabalho que têm em mãos [50], [51].

5.3.2 TEST DRIVEN DEVELOPMENT

O Test Driven Development (TDD) é um processo de desenvolvimento de software em que os testes para o código de produção são criados antes do próprio código de produção. No centro do TDD encontra-se um ciclo de desenvolvimento, tal como mostra a figura 5.5, composto por três fases, com uma ordem específica, que são enumeradas seguidamente:

1. Escrever um teste, que falha e que eventualmente nem compila, para a funcionalidade que se quer implementar.
2. Escrever o código estritamente necessário para fazer o teste passar.
3. Reestruturar (*refactoring*) o código e o teste.

Um dos benefícios de usar TDD é a garantia de que todas as funcionalidades do sistema de informação tem testes que verificam a sua operação. Sempre que é criada uma nova funcionalidade ou

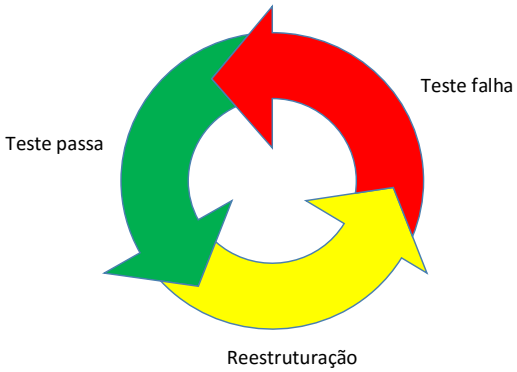

Figura 5.5: Ciclo de desenvolvimento do TDD

é alterada uma funcionalidade existente, o conjunto de testes previamente criados previnem, através da sua execução, que seja introduzido algum comportamento indesejado em alguma funcionalidade existente. Deste modo, quem desenvolve o sistema tem menos receio de introduzir alterações pois é alertado rapidamente caso tenha criado algum problema. Este é um ponto importante pois uma das grandes causas de um sistema ficar num estado em que se torna difícil introduzir novas funcionalidades ou fazer alterações às funcionalidades existentes é o receio que os programadores têm que ao fazer qualquer tipo de alteração para melhorar a estrutura do sistema, este deixe de funcionar devidamente.

Outro beneficio importante do TDD é a criação de código que espelha mais fielmente o intuito pretendido por quem invoca esse código, tornando-se mais fácil de compreender. Ao criar o teste antes de implementar a funcionalidade, o programador assume o ponto de vista de quem vai usar essa funcionalidade e deste modo vai criar uma interface mais clara para quem a vai usar.

A utilização de TDD também contribui para algumas propriedades desejáveis no código de um sistema de informação. Ao criar código devidamente testável para uma funcionalidade, tem que se desacoplar o código do resto do sistema. Um sistema em que os seus componentes estejam pouco acoplados é prenúncio de um sistema bem estruturado e com um bom desenho.

Também se pode argumentar que os testes são uma boa forma de documentação, para os programadores, que é composta por exemplos de como se pode usar o código desenvolvido. Esta documentação, ao contrário de outros tipos de documentação, não corre o risco de ficar desatualizada.

5.4 TECNOLOGIAS USADAS

5.4.1 SPRING

Para desenvolver a API foi usada a *framework* Spring [52]. A escolha da *framework* recaiu sobre o Spring devido, fundamentalmente, às funcionalidades adicionais que esta oferece em relação à implementação de referência da especificação Java API for RESTful Services (JAX-RS) [53], o Jersey [54]. À data da escolha da *framework* para implementar a API, a última versão do Jersey, pronta a ser usada em produção, era a 2.13. Esta versão do Jersey não suporta, nativamente, alguns aspetos

desejados para uma API RESTful, tais como o método HTTP PATCH, o OAuth 2 ou o HAL. Foram analisadas mais algumas *frameworks* para a Java Virtual Machine (JVM), tais como o RestEasy 3.09 e o Dropwizard 0.7.0, mas todas sofriam das mesmas limitações do Jersey, fruto de implementarem a mesma especificação que o Jersey ou até usarem o Jersey. Para a escolha do Spring também foi tido em conta a sua grande aceitação na comunidade de programadores e a quantidade e qualidade da documentação. Outro aspeto relevante na seleção do Spring foi o conhecimento que a equipa de desenvolvimento do PM possui sobre o Spring devido ao seu anterior uso em outras áreas do PM.

Além de terem sido usadas as funcionalidades base do Spring, como é o caso da injeção de dependências, também foram utilizadas as funcionalidades oferecidas pelo Spring MVC e pelo Spring HATEOAS. O Spring MVC é a *framework* do Spring para a camada de apresentação, principalmente quando a camada de apresentação é baseada na *Web*. O Spring HATEOAS oferece funcionalidades para facilitar a implementação, como o nome indica, da restrição HATEOAS do REST. Além destes projetos do Spring, inicialmente recorreu-se ao Spring Boot, para implementar a API, devido à sua capacidade de facilitar a construção da base de uma aplicação. O Spring Boot configura automaticamente alguns aspetos da plataforma Spring que usualmente têm que ser configurados manualmente e também oferece algumas funcionalidades que podem ser úteis para uma aplicação em produção. No entanto, ao automatizar algumas configurações, o Spring Boot acaba por assumir algumas coisas que posteriormente se revelam por não ser o desejado e em alguns casos o esforço que é exigido para alterar esse comportamento acaba por não compensar o uso do Spring Boot. Por esse motivo, o Spring Boot foi abandonado numa fase mais adiantada do desenvolvimento da API quando se começou a implementar o mecanismo de manipulação dos erros da API e se observou que a configuração assumida pelo Spring Boot impedia o funcionamento do mecanismo e o modo de alterar essa configuração, ao contrário do que é normal nos projetos Spring, não se encontrava devidamente documentado.

5.4.2 MAVEN

O Maven é uma ferramenta para gerir projetos Java [55]. Muitas vezes, o Maven é referido e utilizado apenas como uma ferramenta para automatizar a compilação de um projeto mas o Maven oferece muito mais que isso. Além de automatizar a compilação, o Maven também oferece maneiras para ajudar a gerir outros aspetos de um projeto, tais como as dependências ou os artefactos que resultam da compilação.

O Maven foi criado com o intuito de suprimir alguns dos problemas apresentados pelo Ant [56], mas o Maven peca por não oferecer tanta flexibilidade como o Ant oferece. Em contrapartida, o Maven suporta a gestão de dependências, algo que só mais tarde foi introduzido no Ant com o Apache Ivy [57], e permite uma mais fácil compreensão da infraestrutura do projeto através da aplicação de um conjunto de normas. Mais recentemente surgiu o Gradle [58] com o objetivo de ser um meio-termo entre as abordagens do Ant e do Maven. Neste momento, o Gradle já é adotado por grandes nomes da indústria e por projetos importantes tais como o Spring ou Android.

No PM, a ferramenta atualmente utilizada é o Ant mas existem esforços no sentido de se migrar para o Maven. Por esse motivo, a escolha para a API recaiu sobre o Maven e não sobre o Gradle.

5.4.3 SUBVERSION

Um sistema de controlo de versões é um sistema colaborativo que permite manter múltiplas versões de ficheiros através do registo das alterações efetuadas, ao longo do tempo, aos ficheiros. Com um sistema de controlo de versões torna-se possível recuperar versões antigas de um ficheiro ou examinar o histórico das alterações a um ficheiro. É um sistema colaborativo no sentido em que os diferentes elementos de uma equipa de desenvolvimento de software utilizam o mesmo sistema de controlo de versões para manter o código do sistema de informação que estão a desenvolver.

Tal como acontece com a ferramenta de gestão de projeto, o PM também utiliza um sistema de controlo de versões, o Subversion [59], que atualmente já é preterido, pelos programadores, em favor de outro sistema, o Git [60]. No Subversion, os registos de todas as alterações efetuadas aos ficheiros são guardados num repositório. Este repositório tem uma estrutura recomendada assente em diferentes "linhas" de desenvolvimento. O *trunk* é a principal linha de desenvolvimento num repositório e existe desde a origem do projeto que se encontra no repositório. Dependendo do modo de operação, esta linha de desenvolvimento pode ser a mais estável ou a mais instável, sendo que a primeira opção tem ganho primazia nos últimos anos [61]. Do *trunk* usualmente surgem ramificações a que se dá o nome de *branches*. Estes *branches* são cópias do *trunk*, feitas num determinado momento, e normalmente são usadas para desenvolver novas funcionalidades sem comprometer a estabilidade do *trunk*. As funcionalidades implementadas nestes *branches* em alguma altura terão que ficar disponíveis no *trunk*. Para isso é feito um *merge*, que não é mais do que replicar para uma linha de desenvolvimento as alterações feitas numa outra linha de desenvolvimento.

Para a API foi criado, no repositório do PM, um *branch* de desenvolvimento com todos os projetos necessários para a API funcionar. Mais tarde, quando a equipa começar a desenvolver a versão do PM em que será disponibilizada a nova versão da API, será feito o *merge* do *branch* para o *trunk*.

5.4.4 JUNIT

No ecossistema Java, a *framework* mais popular para executar testes de unidade é o JUnit [62]. Uma classe de teste em JUnit é um Plain Old Java Object (POJO) com algumas anotações que configuram a execução dos testes contidos na classe. Na listagem 17 é exibido um teste de unidade em que é testado se o método "getAdaptation", da classe "AdaptationGateway", retorna a devida adaptação quando é fornecido o identificador da adaptação. Pode-se ver que o teste consiste em um método, com a anotação "@Test", onde é invocado o método a testar e é verificado o retorno com recurso a asserções. Uma asserção é uma condição que tem que ser verdadeira para que o teste seja executado com sucesso. Além do método de teste, pode-se ver o método "setUp" que prepara o ambiente de teste e é anotado com "@Before" para indicar que deve ser executado antes de cada método anotado com "@Test". Neste exemplo foram omitidas as declarações dos métodos que são invocados no método "setUp".

```
public class AdaptationGatewayTest {

    private AdaptationGateway adaptationGateway;
    private Adaptation repbssAdaptation;

    @Before
    public void setUp() {
        buildGateway();
        buildRepbssAdaptation();
    }

    @Test
    public void getAdaptation_GivenRepbssId_ShouldReturnRepbssAdaptation() {
        Adaptation adaptationFound = adaptationGateway.getAdaptation(repbssAdaptation.getId());

        assertThat(adaptationFound).isEqualTo(repbssAdaptation);
    }
}
```

Listagem 17: Exemplo de teste de unidade

5.4.5 TRUTH

Truth é uma *framework* de testes que permite criar testes mais fáceis de ler e compreender do que usando apenas a *framework* JUnit [63]. Para isso, o Truth oferece um conjunto de asserções que permitem escrever condições de teste que quase se podem ler como a linguagem natural em inglês. Além disto, as mensagens geradas quando os testes falham também são mais descritivas que as mensagens geradas pelo JUnit. A listagem 18 mostra um excerto de um teste criado exclusivamente com o JUnit e a listagem 19 mostra a mensagem de erro que resulta de executar esse teste. Nas listagens 20 e 21 pode-se visualizar qual o aspeto do mesmo teste e mensagem de erro quando se usa a *framework* Truth. Apenas por este pequeno exemplo já se consegue descortinar as vantagens, que foram enumeradas anteriormente, de usar o Truth.

```
Set<Foo> foo = ...;
assertTrue(foo.isEmpty());
```

Listagem 18: Teste em que não é usada a *framework* Truth

```
java.lang.AssertionError
    at org.junit.Assert.fail(Assert.java:92)
    at org.junit.Assert.assertTrue(Assert.java:43)
    ...
```

Listagem 19: Resultado da execução do teste da listagem 18

```
Set<Foo> foo = ...;
assertThat(foo).isEmpty()
```

Listagem 20: Teste em que é usada a *framework* Truth

```
org.truth0.FailureStrategy$ThrowableAssertionError: Not true that is empty
    at org.truth0.FailureStrategy.fail(FailureStrategy.java:33)
    ...
```

Listagem 21: Resultado da execução do teste da listagem 20

5.4.6 MOCKITO

Quando uma classe a ser testada depende de outras classes, para a testar devidamente é necessário isolar a classe das outras classes que depende. Para este efeito podem-se usar um conjunto de objetos a que se dá o nome de *test doubles*. Um *test double* é um objeto que é usado em vez do objeto real com o intuito de executar um teste. Os *test doubles* podem ser categorizados em tipos diferentes, sendo um deles o *mock*. Um *mock* é um objeto que é configurado para simular o comportamento do objeto real em circunstâncias controladas. O que isto significa na prática é que são declaradas expectativas de serem invocados métodos do *mock* e na execução do teste são comparadas as invocações realmente efetuadas com as que eram esperadas.

Existem várias *frameworks* que auxiliam na criação de *mocks*, sendo a Mockito a mais popular no mundo Java [64].

5.4.7 REST ASSURED

O REST Assured é uma linguagem específica do domínio (DSL - Domain Specific Language) que permite simplificar os testes a uma API RESTful [65]. Uma das funcionalidades oferecidas pelo REST Assured é a verificação da estrutura da representação de um recurso. No caso específico da API, foi criado um ficheiro, para cada representação, onde é descrita a estrutura que a representação pode ter. Para descrever a estrutura foi usado a especificação JSON Schema [66]–[68]. Na listagem 22 pode-se ver o exemplo de uma descrição em JSON Schema de uma representação de uma adaptação.

Na listagem 23 é mostrado um exemplo de um teste feito à API, usando o REST Assured, com o objetivo de verificar se ao fazer um pedido válido para obter uma adaptação específica, a API retorna uma representação válida da adaptação. Além de verificar se a representação é válida, o teste também verifica se o código de estado da resposta é o esperado (200 OK) e se o tipo de média da resposta é o esperado ("application/vnd.nokia-adaptation+json"). Como se pode comprovar, o teste é fácil de ler e compreender graças à expressividade do REST Assured. Para além do método de teste, importa referir que para testar os controladores da API são usados *mocks*, sendo que na listagem 23 o *mock* está a ser criado no método "setUp". Estes *mocks* permitem testar um controlador para além do que um simples teste de unidade poderia testar, porque são tidas em conta as anotações, do Spring, presentes no código. Além disto, em vez do método, a ser testado, ser invocado diretamente pelo teste, como é normal num teste de unidade, o método é invocado através de um *servlet*, tal como acontece quando a API corre no servidor aplicacional. Um *servlet* é um pequeno programa que corre num servidor *Web*, responsável por receber e responder aos pedidos dos clientes *Web*.

```
{
  "title":"Adaptation",
  "description":"A representation of an adaptation",
  "type":"object",
  "properties":{
    "_links":{
      "type":"object",
      "properties":{
        "self":{
          "type":"object",
          "properties":{
            "href":{
              "type":"string"
            }
          }
        }
      },
      "required":[
        "self"
      ]
    },
    "id":{
      "type":"string"
    },
    "description":{
      "type":"string"
    },
    "contentReleases":{
      "type":"array",
      "items":{
        "type":"object",
        "properties":{
          "id":{
            "type":"string"
          },
          "description":{
            "type":"string"
          },
          "technology":{
            "type":"string"
          },
          "group":{
            "type":"string"
          },
          "adaptationSupport":{
            "type":"boolean"
          },
          "version":{
            "type":"string"
          }
        },
        "required":[
          "id",
          "description",
          "technology",
          "group",
          "adaptationSupport",
          "version"
        ]
      }
    }
  },
  "required":[
    "id",
    "description",
    "contentReleases"
  ]
```

Listagem 22: Descrição, em JSON Schema, da representação de uma adaptação

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = ApiConfiguration.class)
@WebAppConfiguration
public class AdaptationRestControllerTest {

    @Autowired
    private WebApplicationContext webApplicationContext;

    @Before
    public void setUp() throws Exception {
        RestAssuredMockMvc.webAppContextSetup(webApplicationContext);
    }

        @Test
        public void getAdaptation_GivenValidAdaptationId_ShouldReturnAdaptationRepresentation() {
            when().
                get("/adaptations/REPBSS").
            then().
                statusCode(HttpStatus.SC_OK).
                contentType("application/vnd.nokia-adaptation+json").
                body(matchesJsonSchemaInClasspath("json/schemas/adaptations/adaptation-schema.json"));
        }
}
```

Listagem 23: Teste para verificar se um pedido válido para obter uma adaptação retorna a representação da adaptação

5.4.8 ASSERTJ

Tal como a *framework* Thruth, o AssertJ oferece um conjunto de asserções mais expressivas que o JUnit. Onde o AssertJ se destaca do Thruth é na simplicidade da criação de asserções personalizadas para as classes que se quer testar. Com isso, pode ser criada uma linguagem, específica para o domínio em causa, que aumenta a expressividade dos testes. Trata-se de uma abordagem que é defendida pelo *Domain Driven Design* para a criação da camada de lógica do domínio de um sistema de informação e que pode ser estendida para a criação dos testes.

5.4.9 SWAGGER

O Swagger é uma *framework* que além de permitir criar a descrição de uma API, também oferece funcionalidades para gerar automaticamente o código, do lado do cliente, responsável por invocar uma API [18]. O Swagger disponibiliza diversas maneiras de criar a descrição de uma API, mas a abordagem mais interessante, e que é adotada, consiste no uso de anotações na API que posteriormente são usadas pelo Swagger para gerar automaticamente a descrição. A descrição gerada pelo Swagger é representada em JSON e pode ser acedida da mesma forma que os recursos da API são acedidos. Significa isto que se o ponto de entrada de uma API for "www.api.com" então a documentação pode ser acedida através de "www.api.com/api-docs". A descrição gerada pelo Swagger também segue a restrição HATEOAS, sendo que em "/api-docs" obtém-se uma descrição geral da API e hiperligações para a descrição de cada recurso suportado pela API. Para além destas funcionalidades, o Swagger ainda disponibiliza um projeto em HTML que acede à descrição de uma API e gera uma página embelezada de documentação onde é possível executar exemplos de pedidos à API [69].

capítulo 6

Conclusão e Trabalho Futuro

6.1 conclusão

Esta dissertação visou o desenvolvimento de uma API RESTful para o sistema de gestão de desempenho de redes de telecomunicações da Nokia. Este sistema, de nome PM, permite que os operadores de telecomunicações analisem o comportamento dos equipamentos de telecomunicações e a eficiência da rede de telecomunicações. O PM insere-se num ecossistema de aplicações desenvolvidas pela Nokia, de suporte à operação de uma rede de telecomunicações, que precisam de comunicar entre si para desempenharem as suas tarefas. Com esse propósito, à muitos anos atrás, foi desenvolvida uma API para o PM que oferece as funcionalidades mais relevantes para as outras aplicações de suporte à operação. Fruto da sua antiguidade e modo de implementação, essa API já não consegue responder às necessidades das aplicações que a usam ou que pretendem usá-la. Por esse motivo, a Nokia decidiu desenvolver uma nova versão da API que servisse não só para as outras aplicações da Nokia acederem ao PM mas também para suportar a evolução do PM para HTML5.

A quantidade de funcionalidades oferecidas pelo PM é bastante extensa e por essa razão foi necessário identificar as funcionalidades essenciais para os clientes e focar o desenvolvimento da API nessas funcionalidades. Trata-se de uma rampa de lançamento para uma API, abrangente, que satisfaça todos os requisitos dos clientes. Esta API é um passo importante para a evolução do PM pois irá permitir não só a evolução do portal *Web* do PM, como também vai melhorar a integração de outros produtos da Nokia com o PM. Mas as vantagens não se ficam só por aqui, pois com a introdução da nova API torna-se possível alterar os serviços do PM sem que os clientes deixem de funcionar corretamente, abrindo assim portas à "limpeza" e evolução destes serviços.

O processo de desenvolvimento da API começou com o estudo do estilo arquitetural REST e de seguida procedeu-se à análise da API existente e levantamento dos requisitos para a nova API. Desta análise resultou um documento com a especificação da nova API, que foi disponibilizado dentro

da Nokia com o objetivo de ser revisto pelas partes interessadas. Após a especificação estabilizar, procedeu-se à seleção das ferramentas a usar para desenvolver a API. Por fim, foi implementada a API de acordo com a especificação elaborada e em concordância com as boas práticas estabelecidas pela Nokia.

O desenvolvimento de uma API RESTful para um sistema complexo, como é exemplo o PM, é um processo longo e sinuoso. Um dos maiores desafios encontrados foi perceber o domínio em que a API se encaixa e os requisitos dos clientes. A área das telecomunicações, por si só, já é bastante complexa e quando se alia os conceitos da gestão de desempenho, ainda mais complexa se torna. Após superar esta dificuldade e se adquirir o conhecimento necessário para se poder começar a desenhar e desenvolver a API, foi encontrado outro obstáculo tão ou mais difícil de transpor que o anterior. Neste caso, a dificuldade encontrada esteve relacionada com a complexidade do projeto PM, derivada, em parte, dos muitos anos que possui de desenvolvimento. Este obstáculo fez-se sentir mais ainda quando se teve que integrar a API com o PM. Pode-se ainda falar de algumas dificuldades, compreensíveis, na adoção do TDD, isto porque se trata de uma maneira completamente diferente de desenvolver software que requer uma mudança de mentalidade.

Devido à API ainda não se encontrar em produção, é impossível avaliar o grau de satisfação dos clientes da API. No entanto, como a API foi desenvolvida em cumprimento com os padrões de qualidade elevados da Nokia e da equipa de desenvolvimento do PM, espera-se que os clientes tenham uma excelente experiência de utilização da API.

Não se podia deixar de referir que a integração numa empresa com o prestígio da Nokia e numa equipa de desenvolvimento com muitos anos de experiência e com elevado nível de conhecimentos permitiu cimentar competências adquiridas ao longo do percurso académico e adquirir novas competências importantes para o futuro profissional.

6.2 TRABALHO FUTURO

Alguns dos obstáculos encontrados, e que foram anteriormente referidos, estão identificados há algum tempo pela equipa de desenvolvimento do PM e já estão planeadas, ou em curso, ações para melhorar algumas áreas. Uma das áreas que está a ser alvo de melhorias é precisamente o mecanismo de compilação do projeto, um dos aspetos mais penosos para alguém que se inicia no desenvolvimento do PM. As melhorias nesta área consistem na migração do projeto PM para Maven, o que provavelmente vai afetar de alguma maneira a API. Outra área onde se está a planear agir é na arquitetura do projeto. Neste momento, existe um grande monólito onde está inserido grande parte do código do PM. A ideia passa por dividir o monólito em vários componentes, faltando definir como vai ser feita essa divisão. Neste caso, a API pode acompanhar a mesma filosofia e ser dividida de acordo com o critério desejado. Um dos possíveis caminhos a seguir é adotar a arquitetura em micro-serviços e dividir o monólito em pequenos serviços, cada um com a sua API RESTful. Esta abordagem traz algumas vantagens, entre as quais se encontra a possibilidade de se usar diferentes tecnologias em cada serviço indo ao encontro das necessidades específicas de cada serviço. Outra vantagem é a possibilidade de o sistema ser mais resiliente, caso tenha sido desenhado com essa preocupação em mente. Os micro-serviços também apresentam uma melhor escalabilidade pois pode-se escalar serviços individualmente de acordo com os seus requisitos de desempenho, ao contrário de um monólito onde se tem que escalar o sistema inteiro. Existe uma vantagem que normalmente é associada aos micro-serviços, que é a facilidade com que se

altera o sistema e se envia as alterações para produção, mas que no caso do PM, devido ao modo como este é disponibilizado para os clientes, esta vantagem não se vislumbra tão facilmente. Finalmente, uma arquitetura em micro-serviços também permite a reutilização e composição de serviços e diminui o custo de substituir um serviço devido à sua reduzida dimensão. As desvantagens associadas aos micro-serviços prendem-se com uma acrescida dificuldade em testar e monitorizar o sistema, juntamente com as dificuldades inerentes a um sistema distribuído como são as transações distribuídas ou o teorema CAP [70].

A migração do PM para HTML5 ainda se encontra numa fase embrionária tendo sido feito até ao momento a migração de uma pequena parte do PM. Está planeado fazer-se a migração de uma parte importante do PM, a visualização dos resultados da execução de um relatório, no ano de 2016. Esta migração irá levar, certamente, ao aparecimento de novos requisitos a implementar na API. Além do HTML5, também é provável o surgimento de outros clientes com requisitos específicos ainda não implementados na API. A segurança da API também é uma área onde ainda existe alguma indefinição, tal como já foi referido, devido à migração, que se vai iniciar brevemente, da plataforma do PM.

Outro trabalho a ser feito é a criação do ambiente de Continuous Integration (CI) para a API. Neste momento já existe uma máquina disponível, com o NetAct instalado, faltando instalar o PM. Após isto, é necessário criar no Jenkins [71] (a ferramenta de CI usada na Nokia) os mecanismos necessários para que o artefacto da API seja construído e instalado no laboratório de cada vez que é feita uma alteração no repositório de código.

Referências

[1] T. Janevski, *NGN Architectures, Protocols and Services*. Hoboken, NJ: John Wiley & Sons, 2014.

[2] Cisco Systems, «Ip telephony: The five nines story», Cisco Systems, rel. téc.

[3] M. Toeroe e F. Tam, *Service Availability: Principles and Practice*. Hoboken, NJ: John Wiley & Sons, 2012.

[4] ITU-T, «Principles for a telecommunications management network», Telecommunication Standardization Sector Of ITU (ITU-T), ITU-T Recommendations M.3010, fev. de 2000.

[5] ——, «Tmn management functions», Telecommunication Standardization Sector Of ITU (ITU-T), ITU-T Recommendations M.3400, fev. de 2000.

[6] R. Fielding e J. Reschke, *Hypertext transfer protocol (http/1.1): message syntax and routing*, RFC 7230 (Proposed Standard), Internet Engineering Task Force, jun. de 2014. endereço: `http://www.ietf.org/rfc/rfc7230.txt`.

[7] ——, *Hypertext transfer protocol (http/1.1): semantics and content*, RFC 7231 (Proposed Standard), Internet Engineering Task Force, jun. de 2014. endereço: `http://www.ietf.org/rfc/rfc7231.txt`.

[8] D. Gourley, B. Totty, M. Sayer, A. Aggarwal e S. Reddy, *HTTP: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, Inc., 2002.

[9] C. Shiflett, *HTTP Developer's Handbook*. Indianapolis, Indiana: Sams Publishing, 2003.

[10] N. Freed e N. Borenstein, *Multipurpose internet mail extensions (mime) part two: media types*, RFC 2046 (Draft Standard), Updated by RFCs 2646, 3798, 5147, 6657, Internet Engineering Task Force, nov. de 1996. endereço: `http://www.ietf.org/rfc/rfc2046.txt`.

[11] M. Belshe, R. Peon e M. Thomson, *Hypertext transfer protocol version 2 (http/2)*, RFC 7540, Internet Engineering Task Force, mai. de 2015. endereço: `http://www.ietf.org/rfc/rfc7540.txt`.

[12] I. Grigorik, *HTTP/2: A New Excerpt from High Performance Browser Networking*. Sebastopol, CA: O'Reilly Media, Inc., 2015.

[13] Google. Spdy, endereço: `https://developers.google.com/speed/spdy/` (acedido em 16/06/2015).

[14] R. Daigneau, *Service Design Patterns*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[15] C. Pautasso, E. Wilde e R. Alarcon, *REST: Advanced Research Topics and Practical Applications*. New York: Springer, 2014.

[16] E. Wilde e C. Pautasso, *REST: From Research to Practice*. New York: Springer, 2011.

[17] R. Fielding, «Architectural styles and the design of network-based software architectures», tese de doutoramento, University of California, Irvine, CA, 2000.

[18] SmartBear. Swagger, endereço: http://swagger.io/ (acedido em 21/05/2015).

[19] U. Sarid, M. Hevery, I. Lazarov, P. Rexer, J. Musser, T. Gullotta, J. Subedar e K. Duffey. Restful api modeling language (RAML), endereço: http://raml.org/ (acedido em 29/06/2015).

[20] Apiary. (2013). Api blueprint, endereço: https://apiblueprint.org/ (acedido em 29/06/2015).

[21] ProgrammableWeb. Programmableweb api directory, endereço: http://www.programmableweb.com/apis/directory (acedido em 29/06/2015).

[22] 3scale. Apis.io, endereço: http://apis.io/ (acedido em 29/06/2015).

[23] ——, Apis.json, endereço: http://apisjson.org/index.html (acedido em 29/06/2015).

[24] M. Amundsen, L. Richardson e M. W. Foster, «Application-level profile semantics (ALPS)», IETF Secretariat, Internet-Draft draft-amundsen-richardson-foster-alps-01, mar. de 2015. endereço: http://www.ietf.org/internet-drafts/draft-amundsen-richardson-foster-alps-01.txt.

[25] M. Foster. (2015). Programming with semantic profiles: In the land of magic strings, the profile-aware is king, endereço: http://www.infoq.com/articles/programming-semantic-profiles (acedido em 08/10/2015).

[26] M. Amundsen, *Building Hypermedia APIs with HTML5 and Node*. Sebastopol, CA: O'Reilly Media, Inc., 2012.

[27] ——, (2014). An interview with hal creator mike kelly, endereço: http://www.infoq.com/articles/web-apis-hal (acedido em 17/09/2015).

[28] S. Klabnik. (2014). Amazon chooses hal media type for appstream api, endereço: http://www.infoq.com/news/2014/03/amazon-hal-appstream (acedido em 17/09/2015).

[29] R. Fielding. (2008). Rest apis must be hypertext-driven, endereço: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (acedido em 30/06/2015).

[30] J. Webber, S. Parastatidis e I. Robinson, *REST in Practice: Hypermedia and Systems Architecture*. Sebastopol, CA: O'Reilly Media, Inc., 2010.

[31] L. Richardson. (2008). Justice will take us millions of intricate moves, endereço: http://www.crummy.com/writing/speaking/2008-QCon/ (acedido em 18/06/2015).

[32] M. Fowler. (2010). Richardson maturity model, endereço: http://martinfowler.com/articles/richardsonMaturityModel.html (acedido em 18/06/2015).

[33] IBM. Websphere application server, endereço: http://www-03.ibm.com/software/products/en/was-overview (acedido em 03/06/2015).

[34] Google. Google json style guide, endereço: https://google-styleguide.googlecode.com/svn/trunk/jsoncstyleguide.xml (acedido em 24/08/2015).

[35] ISO, «ISO 8601:2004», International Organization for Standardization, Geneva, Switzerland, ISO, 2004.

[36] G. Klyne e C. Newman, *Date and time on the internet: timestamps*, RFC 3339 (Proposed Standard), Internet Engineering Task Force, jul. de 2002. endereço: http://www.ietf.org/rfc/rfc3339.txt.

[37] OASIS Security Services (SAML) Technical Committee. Oasis saml wiki, endereço: https://wiki.oasis-open.org/security/FrontPage (acedido em 28/09/2015).

[38] N. Sakimura, J. Bradley, M. B. Jones, B. de Medeiros e C. Mortimore, «Openid connect core 1.0», OpenID Foundation, OpenID Specification, nov. de 2014. endereço: `http://openid.net/specs/openid-connect-core-1_0.html`.

[39] D. Hardt, «The OAuth 2.0 authorization framework», RFC Editor, RFC 6749, out. de 2012. endereço: `http://www.rfc-editor.org/rfc/rfc6749.txt`.

[40] E. Rescorla, *Http over tls*, RFC 2818 (Informational), Updated by RFCs 5785, 7230, Internet Engineering Task Force, mai. de 2000. endereço: `http://www.ietf.org/rfc/rfc2818.txt`.

[41] T. Dierks e C. Allen, *The tls protocol version 1.0*, RFC 2246 (Proposed Standard), Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, Internet Engineering Task Force, jan. de 1999. endereço: `http://www.ietf.org/rfc/rfc2246.txt`.

[42] IETF HTTP Working Group. HTTP/2 frequently asked questions, endereço: `https://http2.github.io/faq/#does-http2-require-encryption` (acedido em 21/09/2015).

[43] D. Stenberg. (2014). HTTP/2 in firefox, endereço: `http://http2-explained.readthedocs.org/en/latest/src/http2firefox.html` (acedido em 21/09/2015).

[44] Open Web Application Security Project (OWASP). (2015). Rest security cheat sheet, endereço: `https://www.owasp.org/index.php/REST_Security_Cheat_Sheet#Authentication_and_session_management` (acedido em 28/09/2015).

[45] R. Fielding e J. Reschke, *Hypertext transfer protocol (http/1.1): authentication*, RFC 7235 (Proposed Standard), Internet Engineering Task Force, jun. de 2014. endereço: `http://www.ietf.org/rfc/rfc7235.txt`.

[46] L. Bass, P. Clements e R. Kazman, *Software Architecture in Practice*, 3ª ed. Upper Saddle River, NJ: Addison-Wesley, 2013.

[47] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord e J. Stafford, *Documenting Software Architectures*, 2ª ed. Upper Saddle River, NJ: Addison-Wesley, 2011.

[48] Mountainminds GmbH & Co. KG and Contributors. (2014). EclEmma, endereço: `http://eclemma.org/` (acedido em 08/10/2015).

[49] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland e D. Thomas. (2001). Manifesto for agile software development, endereço: `http://agilemanifesto.org/` (acedido em 21/05/2015).

[50] Scrum alliance, endereço: `https://www.scrumalliance.org/` (acedido em 21/05/2015).

[51] Scrum.org, endereço: `https://www.scrum.org/` (acedido em 21/05/2015).

[52] Pivotal Software, Inc. Spring, endereço: `http://spring.io/` (acedido em 22/05/2015).

[53] S. Pericas-Geertsen e M. Potociar, *JSR 339: JAX-RS 2.0: The java API for RESTful web services*, Java Specification Requests, 22 de Maio de 2013.

[54] Oracle Corporation. (2010). Jersey, endereço: `https://jersey.java.net` (acedido em 17/09/2015).

[55] The Apache Software Foundation. (2002). Maven, endereço: `http://maven.apache.org/` (acedido em 22/05/2015).

[56] ——, Apache ant, endereço: `http://ant.apache.org/` (acedido em 17/09/2015).

[57] ——, Apache ivy, endereço: `http://ant.apache.org/ivy/` (acedido em 17/09/2015).

[58] Gradle Inc. Gradle, endereço: `https://gradle.org/` (acedido em 17/09/2015).

[59] The Apache Software Foundation. Subversion, endereço: https://subversion.apache.org/ (acedido em 22/05/2015).

[60] Git, endereço: https://git-scm.com/ (acedido em 23/09/2015).

[61] J. Humble e D. Farley, *Continuous Delivery*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[62] Junit, endereço: http://junit.org/ (acedido em 23/09/2015).

[63] Google. Truth, endereço: http://google.github.io/truth/ (acedido em 21/05/2015).

[64] S. Faber, B. Dutheil, I. Czechowski, P. Fornasier, J. Barritt, F. Leipold, L. Keogh, D. North, B. Bańkowski e D. Wallace. Mockito, endereço: http://mockito.org/ (acedido em 22/05/2015).

[65] Jayway. Rest assured, endereço: https://github.com/jayway/rest-assured (acedido em 21/05/2015).

[66] F. Galiegue, K. Zyp e G. Court, «Json schema: Core definitions and terminology», IETF Secretariat, Internet-Draft draft-zyp-json-schema-04, jan. de 2013. endereço: http://www.ietf.org/internet-drafts/draft-zyp-json-schema-04.txt.

[67] K. Zyp e G. Court, «Json schema: Interactive and non interactive validation», IETF Secretariat, Internet-Draft draft-fge-json-schema-validation-00, jan. de 2013. endereço: http://www.ietf.org/internet-drafts/draft-fge-json-schema-validation-00.txt.

[68] G. Luff, K. Zyp e G. Court, «Json hyper-schema: Hypertext definitions for json schema», IETF Secretariat, Internet-Draft draft-luff-json-hyper-schema-00, jan. de 2013. endereço: http://www.ietf.org/internet-drafts/draft-luff-json-hyper-schema-00.txt.

[69] SmartBear. Swagger petstore, endereço: http://petstore.swagger.io/ (acedido em 06/10/2015).

[70] E. Brewer. (2012). Cap twelve years later: How the "rules"have changed, endereço: http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed (acedido em 29/09/2015).

[71] Jenkins, endereço: https://jenkins-ci.org/ (acedido em 02/10/2015).

APÊNDICE A

Documentação da API

A.1 Ponto de entrada

```
https://<subdomain.domain.top-level-domain>/api/performanceManager/v2/
```

a.1.1 Exemplo de interação

```
GET /api/performanceManager/v2
```

```
HTTP 200
Content-Type: application/vnd.nokia-entry-point+json

{
   "_links":{
      "self":{
         "href":"/api/performanceManager/v2"
      },
      "adaptations":{
         "href":"/api/performanceManager/v2/adaptations"
      },
      "report-query-statement":{
         "href":"/api/performanceManager/v2/reports/{report}/query-statement",
         "templated":true
      },
      "report-execution-result":{
         "href":"/api/performanceManager/v2/reports/{report}/execution-result",
         "templated":true
```

```
      },
      "external-object-classes":{
         "href":"/api/performanceManager/v2/external-object-classes{?internalObjectClasses}",
         "templated":true
      },
      "internal-object-classes":{
         "href":"/api/performanceManager/v2/internal-object-classes{?externalObjectClasses}",
         "templated":true
      }
   }
}
```

A.2 VERSÕES

a.2.1 OBTER AS VERSÕES SUPORTADAS

```
GET /api/performanceManager
```

```
HTTP 200
Content-Type: application/vnd.nokia-versioning+json

{
   "_links":{
      "self":{
         "href":"/api/performanceManager/"
      },
      "versions":[
         {
            "href":"/api/performanceManager/v1"
         },
         {
            "href":"/api/performanceManager/v2"
         }
      ]
   }
}
```

A.3 RECURSOS

a.3.1 OBTER ADAPTAÇÕES

DESCRIÇÃO

Lista todas as adaptações instaladas no sistema, incluindo as adaptações combinadas.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-adaptations+json

PEDIDO

```
GET /adaptations
```

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objecto com a chave "adaptations"cujo valor é um array de objetos do tipo adaptation em formato JSON.

EXEMPLO

```
HTTP 200
Content-Type: application/vnd.nokia-adaptations+json

{
   "_links":{
      "self":{
         "href":"/adaptations"
      }
   },
   "adaptations":[
      {
         "_links":{
            "self":{
               "href":"/adaptations/nokacs_content"
            }
         },
         "id":"nokacs_content",
         "description":"nokacs_content Counters and Kpis",
         "contentReleases":[
            {
               "id":"nokacs_content",
               "description":"NSN RACS RS10",
               "group":"Radio",
               "adaptationSupport":false,
               "version":"content"
            }
         ]
      },
      {
         "_links":{
            "self":{
               "href":"/adaptations/RASWPM"
            }
         },
```

```
        "id":"RASWPM",
        "description":"RASWPM Counters and Kpis",
        "contentReleases":[
           {
              "id":"liqapps_content",
              "description":"NSN LiqApps LA15",
              "group":"Radio",
              "adaptationSupport":false,
              "version":"content"
           }
        ]
     }
  ]
}
```

a.3.2 OBTER ADAPTAÇÃO

DESCRIÇÃO

Obtém os detalhes de uma adaptação.

Suporta paginação	Não
Tipo de conteúdo da resposta	application/vnd.nokia-adaptation+json

PEDIDO

```
GET /adaptations/<adaptation>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto do tipo adaptation em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS
```

```
HTTP 200
Content-Type: application/vnd.nokia-adaptation+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS"
      },
      "dimensions":{
         "href":"/adaptations/REPBSS/dimensions{?contentReleases}",
         "templated":true
      },
      "metadata-state":{
         "href":"/adaptations/REPBSS/metadata-state{?timestamp}",
         "templated":true
      },
      "hierarchies":{
         "href":"/adaptations/REPBSS/hierarchies{?dimension,contentRelease}",
         "templated":true
      },
      "measurements":{
         "href":"/adaptations/REPBSS/measurements{?contentRelease,levels}",
         "templated":true
      },
      "kpis":{
         "href":"/adaptations/REPBSS/kpis{?contentRelease,levels}",
         "templated":true
      },
      "levels":{
         "href":"/adaptations/REPBSS/levels{?contentRelease,dimension}",
         "templated":true
      },
      "time-levels":{
         "href":"/adaptations/REPBSS/time-levels{?contentReleases,indicators,reportLevel}",
         "templated":true
      },
      "report-levels":{
         "href":"/adaptations/REPBSS/report-levels{?contentRelease,indicators}",
         "templated":true
      },
      "reports":{
         "href":"/reports{?adaptation,contentReleases,levels,user}",
         "templated":true
      }
   },
   "id":"REPBSS",
   "description":"REPBSS Counters and Kpis",
   "contentReleases":[
      {
         "id":"bss_RG30EP1S3",
         "description":"NSN BSS RG30 EP1 Step3",
```

```
        "group":"Radio",
        "adaptationSupport":false,
        "version":"RG30EP1S3"
      }
   ]
}
```

a.3.3 OBTER O ESTADO DOS META-DADOS DE UMA ADAPTAÇÃO

DESCRIÇÃO

Verifica se o pacote base de conteúdo associado a uma dada adaptação foi atualizado desde uma determinada altura.

Suporta paginação	Não
Tipo de conteúdo da resposta	application/vnd.nokia-adaptation-metadata-state+json

PEDIDO

```
GET /adaptations/<adaptation>/metadata-state?timestamp=<timestamp>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
timestamp	String formatada de acordo com o ISO 8601	O instante que é usado como referencia para comparar e verificar se os meta-dados foram atualizados. Por omissão, é usado o instante atual.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto do tipo adaptation metadata state em formato JSON.

EXEMPLO SEM TIMESTAMP

```
GET /adaptations/nokacs_content/metadata-state
```

```
HTTP 200
Content-Type: application/vnd.nokia-adaptation-metadata-state+json

{
   "_links":{
      "self":{
         "href":"/adaptations/nokacs_content/metadata-state"
      }
   },
   "updated":false
}
```

EXEMPLO COM TIMESTAMP

```
GET /adaptations/nokacs_content/metadata-state?timestamp=2014-02-27T15:05:06+01:00
```

```
HTTP 200
Content-Type: application/vnd.nokia-adaptation-metadata-state+json

{
   "_links":{
      "self":{
         "href":"/adaptations/nokacs_content/metadata_state?timestamp=2014-02-27T15:05:06+01:00"
      }
   },
   "updated":true
}
```

a.3.4 OBTER DIMENSÕES

DESCRIÇÃO

Lista todas as dimensões válidas para uma dada adaptação.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-dimensions+json

PEDIDO

```
GET /adaptations/<adaptation>/dimensions?contentReleases=<content-releases>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-releases	String	Uma lista de identificadores únicos de versões de pacotes de conteúdo, separados por virgulas. Por omissão, são retornados resultados parar todas as versões de pacotes de conteúdo associados à adaptação fornecida.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "dimensions", cujo valor é um array de objetos do tipo dimension em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/dimensions
```

```
HTTP 200
Content-Type: application/vnd.nokia-dimensions+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/dimensions"
      }
   },
   "dimensions":[
      {
         "id":"time",
         "description":"time dimension",
         "isNE":false
      },
      {
         "id":"network_element",
         "description":"network dimension",
         "isNE":true
      }
   ]
}
```

a.3.5 OBTER HIERARQUIAS

DESCRIÇÃO

Lista todas as hierarquias para uma dada adaptação e dimensão.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-hierarchies+json

PEDIDO

```
GET /adaptations/<adaptation>/hierarchies?dimension=<dimension>&contentRelease=<content-release>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
dimension	String	Identificador único que representa uma dimensão especifica. Por omissão, são retornados resultados para a dimensão rede.	Não
content-release	String	Identificador único que representa uma versão de pacote de conteúdo especifica.	Sim

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "hierarquias", cujo valor é um array de objetos do tipo hierarchy em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/hierarchies?contentRelease=bss_RG30EP1S3
```

```
HTTP 200
Content-Type: application/vnd.nokia-hierarchies+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/hierarchies?contentRelease=bss_RG30EP1S3"
      }
   },
   "hierarchies":[
      {
         "id":"PLMN",
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "objectLevels":[
            {
               "id":"PLMN",
               "description":"PLMN",
               "classId":0,
               "isTransient":false,
```

```
            "name":"PLMN",
            "displayName":"PLMN"
         }
      ]
   },
   {
      "id":"PLMN/BSC",
      "dimension":{
         "id":"network_element",
         "description":"network dimension",
         "isNE":true
      },
      "objectLevels":[
         {
            "id":"PLMN",
            "description":"PLMN",
            "classId":0,
            "isTransient":false,
            "name":"PLMN",
            "displayName":"PLMN/BSC"
         },
         {
            "id":"BSC",
            "description":"PLMN/BSC",
            "classId":0,
            "isTransient":false,
            "name":"PLMN/BSC",
            "displayName":"PLMN/BSC"
         }
      ]
   }
]
}
```


a.3.6 OBTER RELATÓRIOS

DESCRIÇÃO

Lista todos os relatórios disponíveis para uma dada adaptação e níveis.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-reports+json

PEDIDO

```
GET /reports?adaptation=<adaptation>&contentReleases=<content-releases>&levels=<levels>&user=<user>
```

PARÂMETROS DO PEDIDO

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
adaptation	String	Identificador único que representa uma adaptação especifica.	Sim
user	String	Identificação do utilizador do qual se quer os relatórios. Por omissão, são retornados os relatórios do utilizador autenticado que faz o pedido.	Não
levels	String	Uma lista de identificadores únicos de níveis dimensionais, separados por virgulas. Por omissão, são retornados resultados para todos os níveis.	Não
content-releases	String	Uma lista de identificadores únicos de versões de pacotes de conteúdo, separados por virgulas. Por omissão, são retornados resultados parar todas as versões de pacotes de conteúdo associados à adaptação fornecida.	Sim

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "reports", cujo valor é um array de objetos do tipo report em formato JSON.

EXEMPLO

```
GET /reports?adaptation=NOKAXC&contentReleases=rsran_RU40
```

```
HTTP 200
Content-Type: application/vnd.nokia-reports+json

{
   "_links":{
      "self":{
         "href":"/reports?adaptation=NOKAXC&contentReleases=rsran_RU40"
      },
      "create-report":{
         "href":"/reports"
      },
      "report-query-statement":{
```

```
      "href":"/reports/{report}/query-statement",
      "templated":true
   },
   "run-report":{
      "href":"/reports/{report}/execution-result",
      "templated":true
   }
},
"reports":[
   {
      "_links":{
         "self":{
            "href":"/reports/PM_10020"
         },
         "delete-report":{
            "href":"/reports/PM_10020"
         }
      },
      "id":"PM_10020",
      "name":"RSRAN144 - End User QOS",
      "creator":"omc",
      "adaptation":{
         "id":"NOKAXC",
         "contentReleases":[
            {
               "id":"rsran_RU40",
               "description":"NSN 3GRAN RU40",
               "group":"Radio",
               "adaptationSupport":false,
               "version":"RU40"
            }
         ],
         "description":"NOKAXC Counters and Kpis"
      },
      "description":"End User QOS",
      "indicators":[
         {
            "_type":"TableColumnGroup",
            "name":"Throughput",
            "children":[
               {
                  "_type":"PI",
                  "id":"AAL2_PATH_GUAR_CELL_RATE",
                  "description":"AAL2_PATH_GUAR_CELL_RATE",
                  "fileName":"rsran/counters/AAL2CAC.xml#AAL2_PATH_GUAR_CELL_RATE"
               }
            ]
         }
      ],
      "dimensionLevels":[
         {
            "id":"day",
            "description":"time level is day",
            "classId":0,
            "isTransient":false,
            "dimension":{
               "id":"time",
```

```
            "description":"time dimension",
            "isNE":false
         },
         "name":"day"
      },
      {
         "id":"PLMN",
         "description":"network level is PLMN",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "name":"PLMN"
      }
   ],
   "timeFilter":{
      "timeType":"RELATIVE",
      "timePeriodLength":1,
      "timeGranularity":"DAY",
      "timePeriodEndType":"LAST_MIDNIGHT"
   },
   "objectInstances":[
   ],
   "isAggregateAware":true,
   "isSharedReport":false,
   "folder":{
      "folderName":"Default",
      "treeDepth":0,
      "level":0,
      "folderMap":[
         {
            "level":0,
            "folder":"../../.."
         }
      ]
   }
},
{
   "id":"rsran_RU40/reports/RSRAN103.xml",
   "name":"RSRAN103 - DSP State Changes",
   "adaptation":{
      "id":"NOKAXC",
      "contentReleases":[
         {
            "id":"rsran_RU40",
            "description":"NSN 3GRAN RU40",
            "group":"Radio",
            "adaptationSupport":false,
            "version":"RU40"
         }
      ],
      "description":"NOKAXC Counters and Kpis"
   },
   "isAggregateAware":true,
```

```
      "isSharedReport":false
   },
   {
      "id":"rsran_RU40/reports/RSRAN047.xml",
      "name":"RSRAN047 - Load Based HO Related Resources",
      "adaptation":{
         "id":"NOKAXC",
         "contentReleases":[
            {
               "id":"rsran_RU40",
               "description":"NSN 3GRAN RU40",
               "group":"Radio",
               "adaptationSupport":false,
               "version":"RU40"
            }
         ],
         "description":"NOKAXC Counters and Kpis"
      },
      "isAggregateAware":true,
      "isSharedReport":false
   },
   {
      "id":"rsran_RU40/reports/RSAXC008.xml",
      "name":"RSAXC008 - AXC ATM VP connection",
      "adaptation":{
         "id":"NOKAXC",
         "contentReleases":[
            {
               "id":"rsran_RU40",
               "description":"NSN 3GRAN RU40",
               "group":"Radio",
               "adaptationSupport":false,
               "version":"RU40"
            }
         ],
         "description":"NOKAXC Counters and Kpis"
      },
      "isAggregateAware":true,
      "isSharedReport":false
   }
  ]
}
```


a.3.7 CRIAR RELATÓRIO

DESCRIÇÃO

Cria um novo relatório com os valores fornecidos.

Tipo de conteúdo do pedido	application/vnd.nokia-report+json
Tipo de conteúdo da resposta	application/vnd.nokia-report+json

PEDIDO

```
POST /reports
Content-Type: application/vnd.nokia-report+json
{
   <report>
}
```

PARÂMETROS DO PEDIDO

Dados presentes no corpo do pedido	
Tipo	Descrição
Objeto report	O corpo do pedido deve conter um objeto do tipo report em formato JSON

RESPOSTA

Em caso de sucesso, a resposta inclui um cabeçalho "Location"com a localização do relatório criado. O corpo da resposta contém um objeto do tipo report em formato JSON.

```
HTTP 201
Content-Type: application/vnd.nokia-report+json
Location: <Url of the new resource that was created by the request>

{
   <report>
}
```

EXEMPLO

```
POST /reports
Content-Type: application/vnd.nokia-report+json

{
   "name":"ReportXpto",
   "adaptation":{
```

```
   "id":"REPBSS",
   "contentReleases":[
      {
         "id":"bss_RG30EP1S3",
         "description":"NSN BSS RG30 EP1 Step3",
         "group":"Radio",
         "adaptationSupport":false,
         "version":"RG30EP1S3"
      }
   ],
   "description":"REPBSS Counters and Kpis"
},
"indicators":[
   {
      "_type":"KPI",
      "id":"pabi_1001a",
      "description":"AVG DL Frame size PA Ethernet average DL Frame size",
      "fileName":"bss_RG30EP1S3/kpis/pabi_1001a.xml#pabi_1001a",
      "formula":"decode(etpebsc.transm_eth_frames_etp_bsc,0,0,etpebsc.transm_eth_frames_etp_bsc)",
      "unit":"byte",
      "format":"float",
      "isCounter":false
   },
   {
      "_type":"KPI",
      "id":"dl_req_egprs_7tsl",
      "description":"DL Request for EGPRS 7 TSL",
      "fileName":"bss_RG30EP1S3/kpis/228_kpis.xml#dl_req_egprs_7tsl",
      "formula":"decode(pcu.req_all_tsl_dl_egprs,0,0,pcu.req_7_tsl_dl_for_egprs_ms)",
      "unit":"%",
      "format":"float",
      "isCounter":false
   }
],
"dimensionLevels":[
   {
      "id":"HOUR",
      "description":"HOUR",
      "classId":0,
      "isTransient":false,
      "dimension":{
         "id":"time",
         "description":"time dimension",
         "isNE":false
      }
   },
   {
      "id":"BSC",
      "description":"PLMN/BSC",
      "classId":0,
      "isTransient":false,
      "dimension":{
         "id":"network_element",
         "description":"network dimension",
         "isNE":true
      }
   }
```

```
   ],
   "timeFilter":{
      "startTime":"2014-12-01T15:05:06+01:00",
      "endTime":"2014-12-02T15:05:06+01:00"
   }
}
```

```
HTTP 201
Content-Type: application/vnd.nokia-report+json
Location: /reports/PM_10060

{
   "_links":{
      "self":{
         "href":"/reports/PM_10060"
      },
      "delete-report":{
         "href":"/reports/PM_10060"
      }
   },
   "id":"PM_10060",
   "name":"ReportXpto",
   "adaptation":{
      "id":"REPBSS",
      "contentReleases":[
         {
            "id":"bss_RG30EP1S3",
            "description":"NSN BSS RG30 EP1 Step3",
            "group":"Radio",
            "adaptationSupport":false,
            "version":"RG30EP1S3"
         }
      ],
      "description":"REPBSS Counters and Kpis"
   },
   "indicators":[
      {
         "_type":"KPI",
         "id":"pabi_1001a",
         "description":"AVG DL Frame size PA Ethernet average DL Frame size",
         "fileName":"bss_RG30EP1S3/kpis/pabi_1001a.xml#pabi_1001a",
         "formula":"decode(etpebsc.transm_eth_frames_etp_bsc,0,0,etpebsc.transm_eth_frames_etp_bsc)",
         "unit":"byte",
         "format":"float",
         "isCounter":false
      },
      {
         "_type":"KPI",
         "id":"dl_req_egprs_7tsl",
         "description":"DL Request for EGPRS 7 TSL",
         "fileName":"bss_RG30EP1S3/kpis/228_kpis.xml#dl_req_egprs_7tsl",
         "formula":"decode(pcu.req_all_tsl_dl_egprs,0,0,pcu.req_7_tsl_dl_for_egprs_ms)",
         "unit":"%",
         "format":"float",
         "isCounter":false
      }
```

```
   ],
   "dimensionLevels":[
      {
         "id":"HOUR",
         "description":"HOUR",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"time",
            "description":"time dimension",
            "isNE":false
         }
      },
      {
         "id":"BSC",
         "description":"PLMN/BSC",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         }
      }
   ],
   "timeFilter":{
      "startTime":"2014-12-01T15:05:06+01:00",
      "endTime":"2014-12-02T15:05:06+01:00"
   }
}
```

a.3.8 APAGAR RELATÓRIO

DESCRIÇÃO

Apaga um relatório.

PEDIDO

```
DELETE /reports/<report>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
report	String	Identificador único que representa um relatório especifico.

RESPOSTA

```
HTTP 204
```

EXEMPLO

```
DELETE /reports/PM_10060
```

```
HTTP 204
```

a.3.9 OBTER INSTRUÇÃO DE EXECUÇÃO DE RELATÓRIO

DESCRIÇÃO

Obtém a instrução SQL que gera os dados para um dado relatório.

Tipo de conteúdo da resposta	application/vnd.nokia-report+json

PEDIDO PARA RELATÓRIO DEFINIDO NO CORPO DO PEDIDO

```
POST /reports/<report>/query-statement
Content-Type: application/vnd.nokia-report+json
Accept: application/vnd.nokia-report+json
X-HTTP-Method-Override: GET

{
   <report>
}
```

PEDIDO PARA RELATÓRIO EXISTENTE NO SISTEMA

```
GET /reports/<report>/query-statement
```

PARÂMETROS DO PEDIDO PARA RELATÓRIO DEFINIDO NO CORPO DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
report	String	Identificador único que representa um relatório específico

Dados presentes no corpo do pedido		
Nome	Tipo	Descrição
report	Objeto report	O corpo do pedido deve conter um objeto do tipo report em formato JSON

PARÂMETROS DO PEDIDO PARA RELATÓRIO EXISTENTE NO SISTEMA

Parâmetros do "path"		
Nome	Tipo	Descrição
report	String	Identificador único que representa um relatório específico

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto do tipo report em formato JSON, que por sua vez contém a chave "queryStatement"cujo valor é um objeto do tipo SQL Statement.

EXEMPLO

```
POST /reports/ReportXpto/query-statement
Content-Type: application/vnd.nokia-report+json
Accept: application/vnd.nokia-report+json
X-HTTP-Method-Override: GET
```

```
{
   "id":"ReportXpto",
   "name":"ReportXpto",
   "adaptation":{
      "id":"REPBSS",
      "contentReleases":[
         {
            "id":"bss_RG30EP1S3",
            "description":"NSN BSS RG30 EP1 Step3",
            "group":"Radio",
            "adaptationSupport":false,
            "version":"RG30EP1S3"
         }
      ],
      "description":"REPBSS Counters and Kpis"
   },
   "indicators":[
      {
         "_type":"KPI",
         "id":"pabi_1001a",
         "description":"AVG DL Frame size PA Ethernet average DL Frame size",
         "fileName":"bss_RG30EP1S3/kpis/pabi_1001a.xml#pabi_1001a",
         "formula":"decode(etpebsc.transm_eth_frames_etp_bsc,0,0,etpebsc.transm_eth_octets_etp_bsc)",
         "unit":"byte",
         "format":"float",
         "isCounter":false
      },
      {
         "_type":"KPI",
         "id":"dl_req_egprs_7tsl",
         "description":"DL Request for EGPRS 7 TSL",
         "fileName":"bss_RG30EP1S3/kpis/228_kpis.xml#dl_req_egprs_7tsl",
         "formula":"decode(pcu.req_all_tsl_dl_egprs,0,0,pcu.req_7_tsl_dl_for_egprs_ms)",
         "unit":"%",
         "format":"float",
         "isCounter":false
      }
   ],
   "dimensionLevels":[
      {
         "id":"HOUR",
         "description":"HOUR",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"time",
            "description":"time dimension",
            "isNE":false
         }
      },
      {
         "id":"BSC",
         "description":"PLMN/BSC",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
```

```
            "description":"network dimension",
            "isNE":true
         }
      }
   ],
   "timeFilter":{
      "startTime":"2014-12-01T15:05:06+01:00",
      "endTime":"2014-12-02T15:05:06+01:00"
   }
}
```

```
HTTP 200
Content-Type: application/vnd.nokia-report+json

{
   "id":"ReportXpto",
   "name":"ReportXpto",
   "adaptation":{
      "id":"REPBSS",
      "contentReleases":[
         {
            "id":"bss_RG30EP1S3",
            "description":"NSN BSS RG30 EP1 Step3",
            "group":"Radio",
            "adaptationSupport":false,
            "version":"RG30EP1S3"
         }
      ],
      "description":"REPBSS Counters and Kpis"
   },
   "indicators":[
      {
         "_type":"KPI",
         "id":"pabi_1001a",
         "description":"AVG DL Frame size PA Ethernet average DL Frame size",
         "fileName":"bss_RG30EP1S3/kpis/pabi_1001a.xml#pabi_1001a",
         "formula":"decode(etpebsc.transm_eth_frames_etp_bsc,0,0,etpebsc.transm_eth_octets_etp_bsc)",
         "unit":"byte",
         "format":"float",
         "isCounter":false
      },
      {
         "_type":"KPI",
         "id":"dl_req_egprs_7tsl",
         "description":"DL Request for EGPRS 7 TSL",
         "fileName":"bss_RG30EP1S3/kpis/228_kpis.xml#dl_req_egprs_7tsl",
         "formula":"decode(pcu.req_all_tsl_dl_egprs,0,0,pcu.req_7_tsl_dl_for_egprs_ms)",
         "unit":"%",
         "format":"float",
         "isCounter":false
      }
   ],
   "dimensionLevels":[
      {
         "id":"HOUR",
         "description":"HOUR",
```

```
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"time",
            "description":"time dimension",
            "isNE":false
         }
      },
      {
         "id":"BSC",
         "description":"PLMN/BSC",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         }
      }
   ],
   "timeFilter":{
      "startTime":"2014-12-01T15:05:06+01:00",
      "endTime":"2014-12-02T15:05:06+01:00"
   },
   "queryStatement":{
      "sql":"select ALLTABLES.period_start_time where ALLTABLES.bsc_gid = bsc.co_gid"
   },
   "isAggregateAware":false,
   "isSharedReport":false
}
```


a.3.10 EXECUTAR RELATÓRIO

DESCRIÇÃO

Obtém os dados resultantes da execução de um relatório fornecido.

Suporta paginação	Não
Tipo de conteúdo da resposta	application/vnd.nokia-report-execution-result+json
Assíncrono	Sim

PEDIDO PARA RELATÓRIO DEFINIDO NO CORPO DO PEDIDO

```
POST /reports/<report>/execution-result
Content-Type: application/vnd.nokia-report+json
Accept: application/vnd.nokia-report-execution-result+json
```

```
X-HTTP-Method-Override: GET

{
   <report>
}
```

PEDIDO PARA RELATÓRIO EXISTENTE NO SISTEMA

```
GET /reports/<report>/execution-result
```

PARÂMETROS DO PEDIDO PARA RELATÓRIO DEFINIDO NO CORPO DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
report	String	Identificador único que representa um relatório especifico

Dados presentes no corpo do pedido		
Nome	Tipo	Descrição
report	Objeto report	O corpo do pedido deve conter um objeto do tipo report em formato JSON

PARÂMETROS DO PEDIDO PARA RELATÓRIO EXISTENTE NO SISTEMA

Parâmetros do "path"		
Nome	Tipo	Descrição
report	String	Identificador único que representa um relatório especifico

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto do tipo SQL Result em formato JSON.

EXEMPLO

```
POST /reports/ReportXpto/execution-result
Content-Type: application/vnd.nokia-report+json
Accept: application/vnd.nokia-report-execution-result+json
X-HTTP-Method-Override: GET

{
   "id":"ReportXpto",
   "name":"ReportXpto",
   "adaptation":{
      "id":"REPBSS",
      "contentReleases":[
         {
            "id":"bss_RG30EP1S3",
            "description":"NSN BSS RG30 EP1 Step3",
            "group":"Radio",
            "adaptationSupport":false,
            "version":"RG30EP1S3"
         }
      ],
      "description":"REPBSS Counters and Kpis"
   },
   "indicators":[
      {
         "_type":"KPI",
         "id":"trf_93",
         "description":"Voice calls on SDCCH",
         "fileName":"bss_RG30EP1S3/kpis/trf_93.xml#trf_93",
         "formula":"resacc.sdcch_emerg_call+resacc.succ_seiz_term",
         "unit":"%",
         "format":"float",
         "isCounter":false
      }
   ],
   "dimensionLevels":[
      {
         "id":"HOUR",
         "description":"HOUR",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"time",
            "description":"time dimension",
            "isNE":false
         },
         "name":"HOUR",
         "displayName":"HOUR"
      },
      {
         "id":"PLMN",
         "description":"PLMN",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
```

```
            "isNE":true
         },
         "name":"PLMN",
         "displayName":"PLMN"
      }
   ],
   "timeFilter":{
      "timeType":"RELATIVE",
      "timePeriodLength":1,
      "timeGranularity":"DAY",
      "timePeriodEndType":"LAST_MIDNIGHT"
   }
}
```

```
HTTP 200
Content-Type: application/vnd.nokia-report-execution-result+json
```


```
{
   "tableModel":{
      "initialRowCapacity":100,
      "initialHeaderRowCapacity":10,
      "strictColumnClass":false,
      "columnCount":6,
      "rowCount":120,
      "headerCount":2,
      "headerColumns":[
         {
            "name":"Period start time",
            "position":0,
            "headerColumns":[

            ]
         },
         {
            "name":"PLMN Name",
            "position":1,
            "headerColumns":[

            ]
         },
         {
            "name":"plmn_gid",
            "position":2,
            "headerColumns":[

            ]
         },
         {
            "name":"DN",
            "position":3,
            "headerColumns":[

            ]
         },
         {
            "name":"Chart Drill",
```

```
            "position":4,
            "headerColumns":[

            ]
         },
         {
            "name":"Voice calls on SDCCH [%]",
            "position":5,
            "headerColumns":[
               {
                  "name":"TRF_93"
               }
            ]
         }
      ],
      "dataRows":[
         [
            "2015-08-14T00:00:00",
            "PLMN",
            "1001",
            "PLMN-PLMN",
            "<img src=\"bss/UI/images/chartIcon.gif\"/>",
            "804.0"
         ],
         [
            "2015-08-14T01:00:00",
            "PLMN",
            "1001",
            "PLMN-PLMN",
            "<img src=\"bss/UI/images/chartIcon.gif\"/>",
            "872.0"
         ],
         [
            "2015-08-14T01:00:00",
            "PLMN",
            "1001",
            "PLMN-PLMN",
            "<img src=\"bss/UI/images/chartIcon.gif\"/>",
            "673.0"
         ],
         [
            "2015-08-14T01:00:00",
            "PLMN",
            "1001",
            "PLMN-PLMN",
            "<img src=\"bss/UI/images/chartIcon.gif\"/>",
            "537.0"
         ]
      ]
   },
   "sqlStatement":{
      "sql":"select resacc.period_start_time where plmn.co_gid in ( '1001' )"
   }
}
```

a.3.11 OBTER MEDIDAS

DESCRIÇÃO

Lista todas as medidas válidas para uma dada adaptação e níveis.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-measurements+json

PEDIDO

```
GET /adaptations/<adaptation>/measurements?contentRelease=<content-release>&levels=<levels>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-release	String	Identificador único que representa uma versão de pacote de conteúdo	Sim
levels	String	Uma lista de identificadores únicos de níveis dimensionais, separados por virgulas. Por omissão, são retornados resultados para todos os níveis.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "measurements", cujo valor é um array de objetos do tipo Measurement em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/measurements?contentRelease=bss_RG30EP1S3
```

```
HTTP 200
Content-Type: application/vnd.nokia-measurements+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/measurements?contentRelease=bss_RG30EP1S3"
      }
   },
   "measurements":[
      {
         "id":"bss_vswr",
         "fileName":"bss/counters/vswr.xml",
         "description":"bss_vswr",
         "counters":[
            {
               "id":"period_duration",
               "description":"period_duration [BSS_VSWR]",
               "fileName":"bss/counters/vswr.xml#period_duration",
               "aggregationFormula":"bss_vswr.period_duration"
            },
            {
               "id":"VSWR",
               "description":"VSWR (C142004) [BSS_VSWR]",
               "fileName":"bss/counters/vswr.xml#VSWR",
               "aggregationFormula":"bss_vswr.vswr"
            },
            {
               "id":"VSWR_SAMPLES",
               "description":"VSWR_SAMPLES (C142005) [BSS_VSWR]",
               "fileName":"bss/counters/vswr.xml#VSWR_SAMPLES",
               "aggregationFormula":"bss_vswr.vswr_samples"
            }
         ],
         "omesName":""
      },
      {
         "id":"mcbsc",
         "fileName":"bss/counters/mcbsc.xml",
         "description":"mcbsc",
         "counters":[
            {
               "id":"ERRORED_ETHERNET_FRAMES",
               "description":"ERRORED_ETHERNET_FRAMES (C134001) [MCBSC]",
               "fileName":"bss/counters/mcbsc.xml#ERRORED_ETHERNET_FRAMES",
               "aggregationFormula":"mcbsc.errored_ethernet_frames"
            },
            {
               "id":"period_duration",
               "description":"period_duration [MCBSC]",
               "fileName":"bss/counters/mcbsc.xml#period_duration",
```

```
            "aggregationFormula":"mcbsc.period_duration"
         },
         {
            "id":"RECEIVED_ETHERNET_FRAMES",
            "description":"RECEIVED_ETHERNET_FRAMES (C134000) [MCBSC]",
            "fileName":"bss/counters/mcbsc.xml#RECEIVED_ETHERNET_FRAMES",
            "aggregationFormula":"mcbsc.received_ethernet_frames"
         },
         {
            "id":"RECEIVED_OCTETS",
            "description":"RECEIVED_OCTETS (C134003) [MCBSC]",
            "fileName":"bss/counters/mcbsc.xml#RECEIVED_OCTETS",
            "aggregationFormula":"mcbsc.received_octets"
         },
         {
            "id":"TRANSMITTED_ETHERNET_FRAMES",
            "description":"TRANSMITTED_ETHERNET_FRAMES (C134002) [MCBSC]",
            "fileName":"bss/counters/mcbsc.xml#TRANSMITTED_ETHERNET_FRAMES",
            "aggregationFormula":"mcbsc.transmitted_ethernet_frames"
         },
         {
            "id":"TRANSMITTED_OCTETS",
            "description":"TRANSMITTED_OCTETS (C134004) [MCBSC]",
            "fileName":"bss/counters/mcbsc.xml#TRANSMITTED_OCTETS",
            "aggregationFormula":"mcbsc.transmitted_octets"
         }
      ],
      "omesName":""
   }
]
}
```


a.3.12 OBTER KPIS

DESCRIÇÃO

Lista todos os KPIs válidos para uma dada adaptação, níveis e utilizador.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-kpis+json

PEDIDO

```
GET /adaptations/<adaptation>/kpis?contentRelease=<content-release>&levels=<levels>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-release	String	Identificador único que representa uma versão de pacote de conteúdo	Sim
levels	String	Uma lista de identificadores únicos de níveis dimensionais, separados por virgulas. Por omissão, são retornados resultados para todos os níveis.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "kpis", cujo valor é um array de objetos do tipo KPI em formato JSON.

EXEMPLO

```
GET /adaptations/NOKLTE/kpis?contentRelease=rslte_RL40
```

```
HTTP 200
Content-Type: application/vnd.nokia-kpis+json

{
   "_links":{
      "self":{
         "href":"/adaptations/NOKLTE/kpis?contentRelease=rslte_RL40"
      },
      "create-kpi":{
         "href":"/adaptations/{adaptation}/kpis",
         "templated":true
      }
   },
   "kpis":[
      {
```

```
            "_links":{
         "self":{
            "href":"/adaptations/NOKLTE/kpis/LTE_765a"
         },
         "delete-kpi":{
            "href":"/adaptations/NOKLTE/kpis/LTE_765a"
         }
      },
      "id":"LTE_765a",
      "description":"E-RAB QCI1releases EPC INI, RNL E-RAB QCI1releases EPC INI due to RNL",
      "fileName":"rslte_RL40/kpis/LTE_765a.xml#LTE_765a",
      "formula":"decode(lepsb.epc_eps_bear_rel_req_n_qci1 + lepsb.epc_eps_bear_rel_req_d_qci1)",
      "unit":"%",
      "format":"float",
      "isCounter":false
   },
   {
      "_links":{
         "self":{
            "href":"/adaptations/NOKLTE/kpis/LTE_530a"
         },
         "delete-kpi":{
            "href":"/adaptations/NOKLTE/kpis/LTE_530a"
         }
      },
      "id":"LTE_530a",
      "description":"UE PWR Headroom PUSCH Lev 30",
      "fileName":"rslte_RL40/kpis/LTE_530a.xml#LTE_530a",
      "formula":"(lpqul.ue_pwr_headroom_pusch_level30)",
      "unit":"#",
      "format":"float",
      "isCounter":false
   },
   {
      "_links":{
         "self":{
            "href":"/adaptations/NOKLTE/kpis/M8001C388"
         },
         "delete-kpi":{
            "href":"/adaptations/NOKLTE/kpis/M8001C388"
         }
      },
      "id":"M8001C388",
      "description":"M8001C388",
      "fileName":"rslte_RL40/kpis/M8001C388.xml#M8001C388",
      "formula":"(lcelld.tb_bund3_nack_pdsch_mcs27)",
      "unit":"#",
      "format":"float",
      "isCounter":false
   }
]
}
```

a.3.13 CRIAR KPI

DESCRIÇÃO

Cria um novo KPI com os valores fornecidos.

Tipo de conteúdo do pedido	application/vnd.nokia-kpi+json
Tipo de conteúdo da resposta	application/vnd.nokia-kpi+json

PEDIDO

```
POST /adaptations/<adaptation>/kpis
Content-Type: application/vnd.nokia-kpi+json
{
   <kpi>
}
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.

Dados presentes no corpo do pedido	
Tipo	Descrição
Objeto kpi	O corpo do pedido deve conter um objeto do tipo kpi em formato JSON

RESPOSTA

Em caso de sucesso, a resposta inclui um cabeçalho "Location"com a localização do KPI criado. O corpo da resposta contém um objeto do tipo kpi em formato JSON.

```
HTTP 201
Content-Type: application/vnd.nokia-kpi+json
Location: <Url of the new resource that was created by the request>
```

```
{
   <kpi>
}
```

EXEMPLO

```
POST /adaptations/NOKLTE/kpis
Content-Type: application/vnd.nokia-kpi+json

{
   "description":"E-RAB QCI1releases EPC INI, RNL E-RAB QCI1releases EPC INI due to RNL",
   "formula":"decode(lepsb.epc_eps_bear_rel_req_n_qci1 + lepsb.epc_eps_bear_rel_req_d_qci1)",
   "unit":"%",
   "format":"float",
   "isCounter":false
}
```

```
HTTP 201
Content-Type: application/vnd.nokia-kpi+json
Location: /adaptations/NOKLTE/kpis/LTE_765a

{
   "_links":{
      "self":{
         "href":"/adaptations/NOKLTE/kpis/LTE_765a"
      },
      "delete-kpi":{
         "href":"/adaptations/NOKLTE/kpis/LTE_765a"
      }
   },
   "id":"LTE_765a",
   "description":"E-RAB QCI1releases EPC INI, RNL E-RAB QCI1releases EPC INI due to RNL",
   "formula":"decode(lepsb.epc_eps_bear_rel_req_n_qci1 + lepsb.epc_eps_bear_rel_req_d_qci1)",
   "fileName":"rslte_RL40/kpis/LTE_765a.xml#LTE_765a",
   "unit":"%",
   "format":"float",
   "isCounter":false
}
```

a.3.14 APAGAR KPI

DESCRIÇÃO

Apaga um KPI.

PEDIDO

```
DELETE /adaptations/<adaptation>/kpis/<kpi>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.
kpi	String	Identificador único que representa um KPI específico.

RESPOSTA

```
HTTP 204
```

EXEMPLO

```
DELETE /adaptations/NOKLTE/kpis/LTE_765a
```

```
HTTP 204
```

a.3.15 OBTER NÍVEIS

DESCRIÇÃO

Lista todos os níveis suportados para uma dada adaptação e dimensão.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-levels+json

PEDIDO

```
GET /adaptations/<adaptation>/levels?contentRelease=<content-release>&dimension=<dimension>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation-id	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-release-id	String	Identificador único que representa uma versão de pacote de conteúdo	Sim
dimension-id	String	Identificador único que representa uma dimensão especifica. Por omissão, são retornados resultados para a dimensão rede	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "levels", cujo valor é um array de objetos do tipo Level em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/levels?contentRelease=bss_RG30EP1S3
```

```
HTTP 200
Content-Type: application/vnd.nokia-levels+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/levels?contentRelease=bss_RG30EP1S3"
      }
   },
```

```
"levels":[
   {
      "_links":{
         "self":{
            "href":"/levels/PLMN"
         },
         "object-instances":{
            "href":"/levels/PLMN/object-instances"
         }
      },
      "id":"PLMN",
      "description":"PLMN",
      "classId":0,
      "isTransient":false,
      "dimension":{
         "id":"network_element",
         "description":"network dimension",
         "isNE":true
      },
      "name":"PLMN",
      "displayName":"PLMN"
   },
   {
      "_links":{
         "self":{
            "href":"/levels/BSS_BAND"
         },
         "object-instances":{
            "href":"/levels/BSS_BAND/object-instances"
         }
      },
      "id":"BSS_BAND",
      "description":"PLMN/BSS_BAND",
      "classId":0,
      "isTransient":false,
      "dimension":{
         "id":"network_element",
         "description":"network dimension",
         "isNE":true
      },
      "name":"PLMN/BSS_BAND",
      "displayName":"PLMN/BAND"
   },
   {
      "_links":{
         "self":{
            "href":"/levels/PSNT"
         },
         "object-instances":{
            "href":"/levels/PSNT/object-instances"
         }
      },
      "id":"PSNT",
      "description":"PLMN/BSC/TCSM/INTF/PSNT",
      "classId":0,
      "isTransient":false,
      "dimension":{
```

```
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "name":"PLMN/BSC/TCSM/INTF/PSNT",
         "displayName":"PLMN/BSC/TCSM/INTF/PSNT"
      }
   ]
}
```

a.3.16 OBTER NÍVEIS COM A DIMENSÃO TEMPO

Lista todos os níveis com a dimensão tempo, para uma dada adaptação.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-levels+json

PEDIDO

```
GET /adaptations/<adaptation>/time-levels?contentReleases=<content-releases>&indicators=<indicators>
        &reportLevel=<report-level>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation-id	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-releases	String	Uma lista de identificadores únicos de versões de pacotes de conteúdo, separados por virgulas. Por omissão, são retornados resultados parar todas as versões de pacotes de conteúdo associados à adaptação fornecida.	Não
report-level	String	Identificador único que representa um nível de rede especifico. Por omissão, são retornados resultados para todos os níveis de rede	Não
indicators	String	Uma lista de identificadores únicos de indicadores, separados por virgulas. Por omissão, são retornados resultados parar todos os indicadores.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "levels", cujo valor é um array de objetos do tipo Level em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/time-levels
```

```
HTTP 200
Content-Type: application/vnd.nokia-levels+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/time-levels"
      }
   },
   "levels":[
      {
         "id":"MONBH",
         "description":"MONBH",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"time",
            "description":"time dimension",
            "isNE":false
         },
```

```
      "name":"MONBH",
      "displayName":"MONBH"
    },
    {
      "id":"QTD",
      "description":"QTD",
      "classId":0,
      "isTransient":false,
      "dimension":{
        "id":"time",
        "description":"time dimension",
        "isNE":false
      },
      "name":"QTD",
      "displayName":"QTD"
    },
    {
      "id":"WEEKBH",
      "description":"WEEKBH",
      "classId":0,
      "isTransient":false,
      "dimension":{
        "id":"time",
        "description":"time dimension",
        "isNE":false
      },
      "name":"WEEKBH",
      "displayName":"WEEKBH"
    }
  ]
}
```


a.3.17 OBTER NÍVEIS COM A DIMENSÃO REDE

Lista todos os níveis com a dimensão rede, para uma dada adaptação.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-levels+json

PEDIDO

```
GET /adaptations/<adaptation>/report-levels?contentRelease=<content-release>&indicators=<indicators>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação especifica.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
content-release	String	Identificador único que representa uma versão de pacote de conteúdo	Sim
indicators	String	Uma lista de identificadores únicos de indicadores, separados por virgulas. Por omissão, são retornados resultados para todos os indicadores.	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "levels", cujo valor é um array de objetos do tipo Level em formato JSON.

EXEMPLO

```
GET /adaptations/REPBSS/report-levels?contentRelease=bss_RG30EP1S3
```

```
HTTP 200
Content-Type: application/vnd.nokia-levels+json

{
   "_links":{
      "self":{
         "href":"/adaptations/REPBSS/report-levels?contentRelease=bss_RG30EP1S3"
      }
   },
   "levels":[
      {
         "id":"PLMN",
         "description":"PLMN",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "name":"PLMN",
```

```
         "displayName":"PLMN"
      },
      {
         "id":"BSC",
         "description":"PLMN/BSC",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "name":"PLMN/BSC",
         "displayName":"PLMN/BSC"
      },
      {
         "id":"BSS_BAND",
         "description":"PLMN/BSS_BAND",
         "classId":0,
         "isTransient":false,
         "dimension":{
            "id":"network_element",
            "description":"network dimension",
            "isNE":true
         },
         "name":"PLMN/BSS_BAND",
         "displayName":"PLMN/BAND"
      }
   ]
}
```


a.3.18 OBTER OBJETOS POR NÍVEL

Lista todos os objetos para um dado nível. Adicionalmente, os resultados podem ser filtrados através de uma "String"de pesquisa

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-objects+json

PEDIDO

```
GET /levels/<level>/object-instances?filter=<filter>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
level	String	Identificador único que representa um nível especifico.

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
filter	String	Usado para filtrar os objetos retornados pela API	Não

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "objects", cujo valor é um array de objetos do tipo Object em formato JSON.

EXEMPLO

```
GET /levels/RNC/object-instances
```

```
HTTP 200
Content-Type: application/vnd.nokia-object-instances+json

{
   "_links":{
      "self":{
         "href":"/levels/RNC/object-instances"
      }
   },
   "objects":[
      {
         "id":"Amadora",
         "globalId":"'9990200000001011197'",
         "name":"RNC",
         "dn":"PLMN-PLMN/RNC-7"
      },
      {
         "id":"Aveiro",
         "globalId":"'9990200000001011117'",
         "name":"RNC",
         "dn":"PLMN-PLMN/RNC-5"
      },
      {
         "id":"Lisbon",
         "globalId":"'9990200000001011157'",
         "name":"RNC",
         "dn":"PLMN-PLMN/RNC-6"
      },
```

```
      {
         "id":"l025v02_RNC-1",
         "globalId":"'999020000001011065'",
         "name":"RNC",
         "dn":"PLMN-PLMN/RNC-1"
      }
   ]
}
```

a.3.19 OBTER CLASSES DE OBJETOS EXTERNAS

Obtém as classes de objetos externas correspondentes às classes de objetos internas fornecidas.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-object-classes+json

PEDIDO

```
GET /external-object-classes?internalObjectClasses=<internal-object-classes>
```

PARÂMETROS DO PEDIDO

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
internal-object-classes	String	Uma lista de identificadores únicos de classes de objetos internas, separados por virgulas	Sim

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "objectClasses", cujo valor é um array de objetos do tipo Object Class em formato JSON.

EXEMPLO

```
GET /external-object-classes?internalObjectClasses=ERTP,ERL
```

```
HTTP 200
Content-Type: application/vnd.nokia-object-classes+json

{
   "_links":{
      "self":{
         "href":"/external-object-classes?internalObjectClasses=ERTP,ERL"
      }
   },
   "objectClasses":[
      {
         "internalId":"ERTP",
         "externalId":"ERTP"
      },
      {
         "internalId":"ERL",
         "externalId":"ERL"
      }
   ]
}
```

a.3.20 OBTER CLASSES DE OBJETOS INTERNAS

Obtém as classes de objetos internas correspondentes às classes de objetos externas fornecidas.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-object-classes+json

PEDIDO

```
GET /internal-object-classes?externalObjectClasses=<external-object-classes>
```

PARÂMETROS DO PEDIDO

Parâmetros da "query"			
Nome	Tipo	Descrição	Obrigatório
external-object-classes	String	Uma lista de identificadores únicos de classes de objetos externas, separados por virgulas	Sim

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "objectClasses", cujo valor é um array de objetos do tipo Object Class em formato JSON.

EXEMPLO

```
GET /internal-object-classes?externalObjectClasses=ERTP,ERL
```

```
HTTP 200
Content-Type: application/vnd.nokia-object-classes+json

{
   "_links":{
      "self":{
         "href":"/internal-object-classes?externalObjectClasses=ERTP,ERL"
      }
   },
   "objectClasses":[
      {
         "internalId":"ERTP",
         "externalId":"ERTP"
      },
      {
         "internalId":"ERL",
         "externalId":"ERL"
      }
   ]
}
```

a.3.21 OBTER LIMIARES

DESCRIÇÃO

Obtém todos os limiares associados a um dado relatório.

Suporta paginação	Sim
Tipo de conteúdo da resposta	application/vnd.nokia-thresholds+json

PEDIDO

```
GET /adaptations/<adaptation>/reports/<report>/thresholds
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.
report	String	Identificador único que representa um relatório específico.

RESPOSTA

Em caso de sucesso, o corpo da resposta contém um objeto com a chave "thresholds", cujo valor é um array de objetos do tipo Threshold em formato JSON.

EXEMPLO

```
GET /adaptations/NOKLTE/reports/ThresholdReport/thresholds
```

```
HTTP 200
Content-Type: application/vnd.nokia-thresholds+json

{
   "_links":{
      "self":{
         "href":"/adaptations/NOKLTE/reports/ThresholdReport/thresholds"
      }
   },
   "thresholds":[
      {
         "_links":{
            "self":{
               "href":"/adaptations/NOKLTE/reports/ThresholdReport/thresholds/3"
            },
```

```
            "delete-threshold":{
               "href":"/adaptations/NOKLTE/reports/ThresholdReport/thresholds/3"
            }
         },
         "id":3,
         "formulaGroupName":"ThresholdName",
         "notificationId":"69032",
         "notificationText":"notificationText1418902411751",
         "formulas":[
            {
               "severity":"WARNING",
               "formula":"${CHNG_TO_CELL_PLAN_UNAVAIL} >= 0"
            },
            {
               "severity":"CRITICAL",
               "formula":"${DENOM_CELL_AVAIL} < 4000"
            },
            {
               "severity":"MAJOR",
               "formula":" ${period_duration_sum} != 0"
            },
            {
               "severity":"MINOR",
               "formula":"${SAMPLES_CELL_AVAIL} = 30000"
            }
         ],
         "requiredSpikes":2,
         "spikeWindow":4,
         "period":1,
         "notifications":[
            {
               "target":"EMAIL",
               "parameters":[
                  {
                     "value":"jose.pinto@nsn.com"
                  },
                  {
                     "value":"jose.pinto@nsn.com"
                  }
               ]
            },
            {
               "target":"SMS",
               "parameters":[
                  {
                     "value":"963333333"
                  },
                  {
                     "value":"969999999"
                  }
               ]
            }
         ]
      }
   ]
}
```

a.3.22 CRIAR LIMIAR

DESCRIÇÃO

Cria um novo limiar com os valores fornecidos.

Tipo de conteúdo do pedido	application/vnd.nokia-threshold+json
Tipo de conteúdo da resposta	application/vnd.nokia-threshold+json

PEDIDO

```
POST /adaptations/<adaptation>/reports/<report>/thresholds
Content-Type: application/vnd.nokia-threshold+json

{
   <threshold>
}
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.
report	String	Identificador único que representa um relatório específico.

Dados presentes no corpo do pedido	
Tipo	Descrição
Objeto threshold	O corpo do pedido deve conter um objeto do tipo threshold em formato JSON

RESPOSTA

Em caso de sucesso, a resposta inclui um cabeçalho "Location"com a localização do limiar criado. O corpo da resposta contém um objeto do tipo threshold em formato JSON.

```
HTTP 201
Content-Type: application/vnd.nokia-threshold+json
Location: <Url of the new resource that was created by the request>

{
   <threshold>
}
```

EXEMPLO

```
POST /adaptations/NOKLTE/reports/ThresholdReport/thresholds
Content-Type: application/vnd.nokia-threshold+json

{
   "formulaGroupName":"ThresholdName",
   "notificationId":"69032",
   "notificationText":"notificationText1418902411751",
   "formulas":[
      {
         "severity":"WARNING",
         "formula":"${CHNG_TO_CELL_PLAN_UNAVAIL} >= 0"
      },
      {
         "severity":"CRITICAL",
         "formula":"${DENOM_CELL_AVAIL} < 4000"
      },
      {
         "severity":"MAJOR",
         "formula":" ${period_duration_sum} != 0"
      },
      {
         "severity":"MINOR",
         "formula":"${SAMPLES_CELL_AVAIL} = 30000"
      }
   ],
   "requiredSpikes":2,
   "spikeWindow":4,
   "period":1,
   "notifications":[
      {
         "target":"EMAIL",
         "parameters":[
            {
               "value":"jose.pinto@nsn.com"
            },
            {
               "value":"jose.pinto@nsn.com"
            }
         ]
      },
      {
         "target":"SMS",
         "parameters":[
```

```
            {
               "value":"963333333"
            },
            {
               "value":"969999999"
            }
         ]
      }
   ]
}
```

```
HTTP 201
Content-Type: application/vnd.nokia-threshold+json
Location: /adaptations/NOKLTE/reports/ThresholdReport/thresholds/3

{
   "_links":{
      "self":{
         "href":"/adaptations/NOKLTE/reports/ThresholdReport/thresholds/3"
      },
      "delete-threshold":{
         "href":"/adaptations/NOKLTE/reports/ThresholdReport/thresholds/3"
      }
   },
   "id":3,
   "formulaGroupName":"ThresholdName",
   "notificationId":"69032",
   "notificationText":"notificationText1418902411751",
   "formulas":[
      {
         "severity":"WARNING",
         "formula":"${CHNG_TO_CELL_PLAN_UNAVAIL} >= 0"
      },
      {
         "severity":"CRITICAL",
         "formula":"${DENOM_CELL_AVAIL} < 4000"
      },
      {
         "severity":"MAJOR",
         "formula":" ${period_duration_sum} != 0"
      },
      {
         "severity":"MINOR",
         "formula":"${SAMPLES_CELL_AVAIL} = 30000"
      }
   ],
   "requiredSpikes":2,
   "spikeWindow":4,
   "period":1,
   "notifications":[
      {
         "target":"EMAIL",
         "parameters":[
            {
               "value":"jose.pinto@nsn.com"
            },
```

```
            {
               "value":"jose.pinto@nsn.com"
            }
         ]
      },
      {
         "target":"SMS",
         "parameters":[
            {
               "value":"963333333"
            },
            {
               "value":"969999999"
            }
         ]
      }
   ]
}
```

a.3.23 APAGAR LIMIAR

DESCRIÇÃO

Apaga um determinado limiar.

PEDIDO

```
DELETE /adaptations/<adaptation>/reports/<report>/thresholds/<threshold>
```

PARÂMETROS DO PEDIDO

Parâmetros do "path"		
Nome	Tipo	Descrição
adaptation	String	Identificador único que representa uma adaptação específica.
report	String	Identificador único que representa um relatório específico.
threshold	String	Identificador único que representa um limiar específico.

RESPOSTA

```
HTTP 204
```

EXEMPLO

```
DELETE /adaptations/NOKLTE/reports/ThresholdReport/thresholds/3
```

```
HTTP 204
```

APÊNDICE B

Recursos da API

Método	**URL**	**Uso**
GET	/adaptations	Obter adaptações
GET	/adaptations/{adaptation}	Obter adaptação
GET	/adaptations/{adaptation}/metadata-state	Obter o estado dos meta-dados de uma adaptação
GET	/adaptations/{adaptation}/dimensions	Obter dimensões
GET	/adaptations/{adaptation}/hierarchies	Obter hierarquias
GET	/reports	Obter relatórios
POST	/reports	Criar relatório
DELETE	/reports/{report}	Apagar relatório
POST	/reports/{report}/query-statement	Obter instrução SQL
POST	/reports/{report}/execution-result	Executar relatório
GET	/adaptations/{adaptation}/measurements	Obter medidas
GET	/adaptations/{adaptation}/kpis	Obter KPIs
POST	/adaptations/{adaptation}/kpis	Criar KPI
DELETE	/adaptations/{adaptation}/kpis/{kpi}	Apagar KPI

Tabela B.1: Recursos da API

Método	**URL**	**Uso**
GET	/adaptations/{adaptation}/levels	Obter níveis
GET	/adaptations/{adaptation}/time-levels	Obter níveis com a dimensão tempo
GET	/adaptations/{adaptation}/report-levels	Obter níveis com a dimensão rede
GET	/levels/{level}/object-instances	Obter objetos por nível
GET	/external-object-classes	Obter classes externas de objetos
GET	/internal-object-classes	Obter classes internas de objetos
GET	/thresholds	Obter limiares
POST	/thresholds	Criar limiar
DELETE	/thresholds/{threshold}	Apagar limiar

Tabela B.2: Recursos da API (Continuação)

Método	**URL**	**Tipo de média**
GET	/adaptations	application/vnd.nokia-adaptations+json
GET	/adaptations/{adaptation}	application/vnd.nokia-adaptation+json
GET	/adaptations/{adaptation}/metadata-state	application/vnd.nokia-adaptation-metadata-state+json
GET	/adaptations/{adaptation}/dimensions	application/vnd.nokia-dimensions+json
GET	/adaptations/{adaptation}/hierarchies	application/vnd.nokia-hierarchies+json
GET	/reports	application/vnd.nokia-reports+json
POST	/reports	application/vnd.nokia-report+json
DELETE	/reports/{report}	-
POST	/reports/{report}/query-statement	application/vnd.nokia-report+json
POST	/reports/{report}/execution-result	application/vnd.nokia-report-execution-result+json
GET	/adaptations/{adaptation}/measurements	application/vnd.nokia-measurements+json
GET	/adaptations/{adaptation}/kpis	application/vnd.nokia-kpis+json
POST	/adaptations/{adaptation}/kpis	application/vnd.nokia-kpi+json
DELETE	/adaptations/{adaptation}/kpis/{kpi}	-

Tabela B.3: Tipos de média dos recursos da API

Método	**URL**	**Tipo de média**
GET	/adaptations/{adaptation}/levels	application/vnd.nokia-levels+json
GET	/adaptations/{adaptation}/time-levels	application/vnd.nokia-levels+json
GET	/adaptations/{adaptation}/report-levels	application/vnd.nokia-levels+json
GET	/levels/{level}/object-instances	application/vnd.nokia-object-instances+json
GET	/external-object-classes	application/vnd.nokia-object-classes+json
GET	/internal-object-classes	application/vnd.nokia-object-classes+json
GET	/thresholds	application/vnd.nokia-thresholds+json
POST	/thresholds	application/vnd.nokia-threshold+json
DELETE	/thresholds/{threshold}	-

Tabela B.4: Tipos de média dos recursos da API (Continuação)