ISEL – Instituto Superior de Engenharia de Lisboa

DEETC – Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Telco Smart Sites – TMForum Standard Translator

Celfocus

CARLOS MIGUEL B. F. VIEGAS
(Licenciado em Engenharia Informática e Multimédia)

Trabalho Final de Mestrado para obtenção do grau de
Mestre em Engenharia Informática e Multimédia

Orientadores:
Prof. Doutor Carlos Jorge de Sousa Gonçalves
Mestre Paulo Alexandre Henriques Patrício

Jurí:
Presidente:
Prof. Doutor Pedro Viçoso Fazenda
Vogais:
Prof. Doutor José Manuel Garcia Simão
Prof. Doutor Carlos Jorge de Sousa Gonçalves

Novembro de 2022

Resumo

Hoje em dia, a Internet das Coisas tem sido um tema cada vez mais importante nas nossas vidas, tendo vindo a proporcionar mudanças na sociedade, na economia e na vida pessoal, mais especificamente na vida nas cidades e na saúde, e em contexto empresarial, pode contribuir para novos meios de realizar vendas, ampliar as habilidades profissionais e auxiliar empreendedores nas suas tomadas de decisões.

Por essa razão, cada vez mais empresas como a Celfocus têm investido nesta área, o que faz com que novas plataformas de Internet das Coisas surjam, cada uma com as suas características e estruturas distintas, mas que podem trabalham com os mesmos tipos de dados, oferecendo ao utilizador funcionalidades diferentes. Apesar de oferecerem funcionalidades diferentes, estas plataformas podem servir de ferramentas potentes para uma empresa quando utilizadas em simultâneo e para o mesmo fim sendo que, em contrapartida a esse ideal, existe uma incompatibilidade entre o mesmo tipo de dados usados por plataformas diferentes por utilizarem estruturas diferentes para os representarem.

É neste contexto que surge o tema da estandardização de dados, na área da Internet das Coisas, aplicado neste projeto através do desenvolvimento de um tradutor que, em fase inicial, permite a tradução de dados da IoT.nxt, uma das plataformas utilizadas pela Celfocus, para uma estrutura de dados definida pelo TMForum, uma organização que representa uma aliança entre diversas outras empresas.

Assim, a estrutura utilizada pelos dados traduzidos é considerada como a linguagem de representação de dados comum entre todas as plataformas que utilizem este método de tradução no futuro, fazendo com que o mesmo tipo de dados provenientes de plataformas diferentes sejam representados da mesma forma, permitindo assim usufruir das potencialidades de cada plataforma em simultâneo.

Palavras-chave: Celfocus, Internet das Coisas, Plataformas, Estandardização, Tradução, TMForum, IoT.nxt.

Abstract

Nowadays, the Internet of Things has been an increasingly important topic in our lives, having brought changes in our society, in the economy and in our personal lives, more specifically in the context of life in the city and health health, and in a business context, can contribute to new ways to increase sales, expand professional skills and help entrepreneurs in their decision making.

For this reason, more and more companies like Celfocus are investing in this area, which makes new Internet of Things platforms emerge, each with its own distinct characteristics and structures, but which can work with the same types of data, offering the user different functionalities. Despite offering different features, these platforms can serve as powerful tools for a company when used at the same time and for the same purpose. The obstacle to this ideal is an incompatibility between the same type of data used by different platforms because they use different structures to represent them.

It's in this context that the issue of data standardization arises, in area of Internet of Things, applied in this project through the development of a translator that, in an initial phase, allows the translation of data fromIoT.nxt, one of the platforms used by Celfocus, to a new data structure defined by TMForum, an organization that represents an alliance between several other companies.

Therefore, the structure used by the translated data is considered as the representation language of data common between all platforms that will use this method of translation in the future, causing the same type of data that comes from different platforms to be represented in the same way, allowing the companies to take advantage of the potential of each platform at the same time.

Keywords: Celfocus, Internet of Things, Platforms, Standardization, Translation tion, TMForum, IoT.nxt.

Agradecimentos

Presto os meus agradecimentos aos meus orientadores Prof. Doutor Carlos Gonçalves, da parte do Instituto Superior de Engenharia de Lisboa, e Mestre Paulo Patrício, da parte da Celfocus, pelo tempo, pela compreensão, pela paciência e por estarem ativamente disponíveis para ajudar com quaisquer problemas que foram ocorrendo, guiando-me da melhor forma possível para a conclusão deste projeto e tornando esta experiência muito mais enriquecedora.

Chegando a esta fase do meu percurso académico, fulcral também agradecer ao ISEL, uma instituição que me acolheu desde a licenciatura até ao fim do mestrado, nomeadamente a todo o Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores, que através dos docentes das diversas unidades curriculares que frequentei, dos colegas de curso com que me cruzei e de todas condições que me foram proporcionadas, contribuiu para o meu crescimento, tanto a nível pessoal como profissional, que pelo conhecimento e experiência que pude adquirir ao longo dos anos permitiu que obtivesse assim um contributo valioso para o desenvolvimento deste projeto.

Quero agradecer à Celfocus pela oportunidade e pelo voto de confiança que levaram à minha participação no estágio e neste projeto, abrindo assim as portas para à minha entrada no mundo profissional.

Finalmente, sinto uma enorme gratidão pelos meus pais, Amadeu e Berta, pelos meus irmãos, Amadina, Bruno e Leandro, e pela minha namorada, Mariana Mendes, por todo o apoio, atenção, preocupação e paciência durante o desenvolvimento do projeto e durante todo o meu percurso académico, tendo sido uma grande fonte de ajuda e motivação que me permitiram alcançar os meus objetivos.

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

Índice de Códigos

Lista de Acrónimos

API *Application Programming Interface*

CoAP *Constrained Application Protocol*

HTTP *Hypertext Transfer Protocol*

IDE *Integrated Development Environment*

IIoT *Industrial Internet of Things*

IoT *Internet of Things*

ISEL Instituto Superior de Engenharia de Lisboa

JSON *JavaScript Object Notation*

MPN *Mobile Private Networks*

MQTT *Message Queuing Telemetry Transport*

NFC *Near Field Communication*

OTA *Over-The-Air*

REST *Representational State Transfer*

Telco *Telecommunications Company*

TMForum *TeleManagement Forum*

URI *Uniform Resource Identifier*

Capítulo 1

Introdução

Nos últimos anos, a *Internet of Things* (IoT), em português Internet das Coisas, tem ganho um aumento enorme de popularidade devido às características que apresenta, tendo atraído novos consumidores, nomeadamente pela sua capacidade de aumentar a produtividade, manter a eficiência e produzir resultados rapidamente.

A IoT permite uma análise e gestão de dados, tornando possível tomadas de decisões inteligentes em tempo real fazendo com que se tenha tornado numa aposta com um grande potencial para indústrias de todos os tipos.

A Celfocus [2], a empresa onde foi realizado o estágio de onde surgiu o projeto em questão, tem estado muito ativa nesta área na Europa, tendo já desenvolvido diversos projetos de múltiplos verticais, ou seja, projetos aplicados em diferentes setores como a da saúde, energético, empresarial, entre outros. No vertical de *Smart Cities*, A equipa de IoT desenvolveu o projeto *Green City*, uma solução para análise de dados em tempo real de problemas de poluição em centros urbanos que levam a problemas de saúde, depreciação de imóveis e baixa qualidade de vida, isto através de

sensores que medem a qualidade do ar. No vertical do *Building Management*, foi desenvolvido o *Meeting Room*, uma solução que permite que as empresas e os seus colaboradores otimizem a utilização das salas de reuniões em tempo real, mantendo registos históricos, permitindo a realização de reservas ou a reportação de problemas.

Neste projeto pretende-se a exploração de uma *framework* para permitir a tradução dos dados criados em diferentes plataformas para a criação de soluções IoT, recorrendo às suas API nativas, para um formato uniforme entre diferentes fontes de dados.

1.1 Objetivos

A solução deste problema será obtida desenvolvendo uma *framework* que será executada recorrendo a um conjunto de várias API do grupo TeleManagement Forum (TMForum) [3].

Ao desenvolver uma solução IoT é possível realizar a integração de várias API, cada uma com uma linguagem própria da plataforma. Por exemplo, para quando há necessidade de expor o conteúdo destas API para sistemas externos, haverão conflitos devido ao formato dos dados que serão enviados, pois cada plataforma terá a sua própria maneira de representar esses dados. Visto que num projeto a plataforma que é usada poderá não ser estável, de forma a que os mesmos sistemas de destino possam receber dados independentemente da fonte, há necessidade de estandardizar o formato dos dados.

Com este projeto pretende-se então o desenvolvimento de um intermediário entre as plataforma IoT e sistemas terceiros que irá funcionar como um tradutor do conteúdo das API utilizadas na plataforma para a linguagem TMForum (TMF) [3], sendo esta última a linguagem que será considerada “universal” pois estes sistemas apenas terão de lidar com essa linguagem.

Inicialmente o conector será preparado para uniformizar dados relativos a alarmes, inventário de recursos e informações geográficas seguindo então as estruturas e regras estabelecidas nas API *Alarm Management* [4], Resource Inventory Management [1] e Geographic Site Management [5], respetivamente, cuja utilidade será descrita na secção 2.2.7.

1.2 Planeamento

Numa fase inicial do estágio, tendo sido definidos os objetivos supracitados, foi organizado um plano com a equipa de IoT com a ordem de tarefas a serem completadas. A Figura 1.1 apresenta cada uma das fases definidas para o desenvolvimento deste projeto.

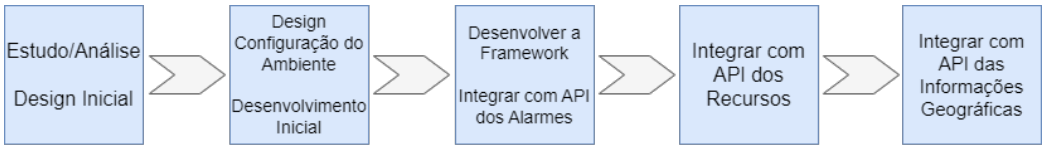

Figura 1.1: Planeamento das fases de desenvolvimento do Projeto.

Durante o processo desenvolvimento, há possibilidade da ocorrência de mudanças no plano, mas ao haver uma base, acaba por ser mais simples organizar o trabalho desenvolvido ao longo tempo e ter sempre noção do progresso realizado.

1.3 Resultados do Projeto

O projeto desenvolvido no contexto deste trabalho irá produzir um conjunto de componentes que podem ser utilizados em projetos internos do passado e futuro da Celfocus, ou seja, este projeto é considerado como

algo com utilidade prática pois o seu resultado será integrado num conjunto de ferramentas da empresa, tornando-se assim um facilitador para resolver necessidades já sentidas anteriormente.

1.4 Organização do Relatório

O objetivo deste relatório é apresentar detalhadamente o desenvolvimento deste TMForum Standard Translator no que toca ao estudo e pesquisa da informação que foi necessária para a sua realização, o processo de implementação com base no conhecimento obtido e o relato das validações e testes após a sua conclusão.

O resto do relatório está organizado em seis partes. As quatro primeiras partes dizem respeito ao resumo dos objetivos e contextualização do projeto, capítulo de Introdução 1, ao estado atual de conhecimento sobre os temas abordados para o desenvolvimento que estão sob estudo e análise, capítulo de Trabalho Relacionado 2, a visualização abstrata das arquiteturas propostas, capítulo de Análise 3 e, finalmente, a sua implementação, capítulo de Implementação do Modelo 4. As duas partes seguintes, numa visão em que o projeto já se encontra desenvolvido, referem-se à colocação do projeto à prova, capítulo de Validação e Testes 5, conclusões e a sua visão futura com base no que já foi alcançado, capítulo de Conclusões e Trabalho Futuro 6.

Capítulo 2

Trabalho Relacionado

Neste capítulo são apresentados os conceitos que serviram para este projeto, nomeadamente a IoT e a área principal para o desenvolvimento deste projeto, a estandardização de Soluções IoT na área das Telecomunicações. Na secção 2.1 é apresentada uma visão geral sobre a *Internet of Things*, onde mais especificamente na secção 2.1.1 serão apresentados detalhes sobre o seu funcionamento, na secção 2.2 são apresentados alguns problemas, vantagens e pressupostos nas sobre o tema da estandardização em soluções IoT e onde também é apresentada a plataforma que irá ser utilizada neste projeto assim como é feita uma comparação com outras disponíveis.

2.1 *Internet of Things*

A *Internet of Things* é um paradigma emergente que possibilita a ligação entre dispositivos e aplicações para tratar de casos de uso, e que esta

ligação pode ser local, por exemplo através do *Wi-Fi*, privada, como por exemplo as Mobile Private Networks (MPN) ou via *Internet*, por exemplo Mobile, mas sempre garantindo a segurança. Através de uma rede de comunicação, usa dispositivos inteligentes e Internet para fornecer soluções inovadoras para vários desafios e questões relacionadas com vários negócios. De forma geral, a IoT é uma inovação que reúne ampla variedade de sistemas, estruturas, dispositivos, sensores e atuadores inteligentes que comunicam entre si de tal forma que já estão parcialmente ou totalmente envolvidas em atividades humanas. A Figura 2.1 apresenta a grande abrangência diferentes dispositivos, entidades e tecnologias que podem ser envolvidos numa só rede.

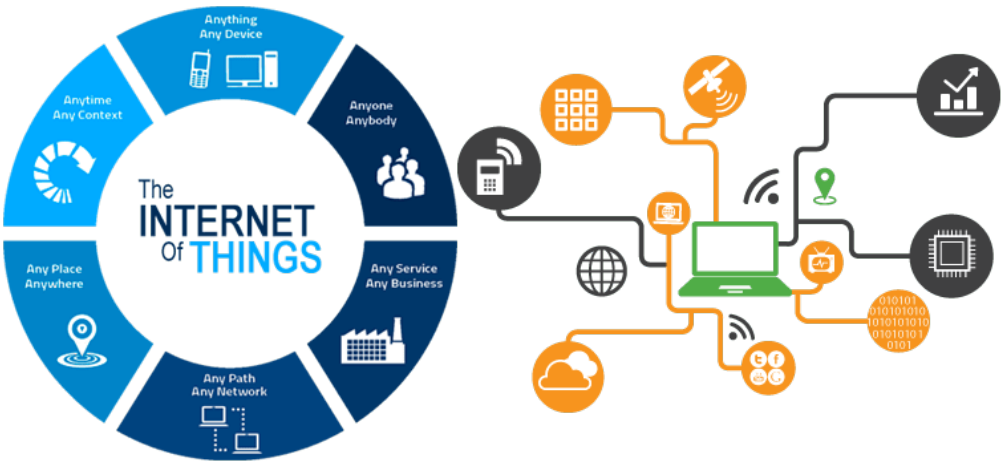

Figura 2.1: Representação de uma rede IoT [6]

O avanço da tecnologia da IoT é bastante evidente ao ponto de atualmente, tal como apresentado na Figura 2.1, existirem formas de conectar praticamente qualquer dispositivo, qualquer negócio, qualquer pessoa e em qualquer lugar, tudo através de uma rede controlada a partir de um só computador, por exemplo.

2.1.1 Funcionamento da IoT

Tomando como base a introdução apresentada na secção 2.1, essencialmente os dispositivos que são utilizados em soluções IoT possuem *hardware* como sensores, por exemplo, que capturam dados que poderão então ser importados para uma plataforma. A partir daí, o plataforma em questão poderá ter como função a análise, tratamento e envio os dados aos utilizadores através de um portal *web* ou através de uma aplicação móvel, por exemplo.

Tal como existem os sensores, em IoT temos também atuadores, que após análise dos dados e regras de negócio atuam fisicamente, por exemplo, torna-se possível desligar a energia caso haja uma inundação em casa, rega de jardins quando não chove ou quando um terreno precisa, abrir uma porta após a alguém ter tocada à campainha ou passado um dispositivo com Near Field Communication (NFC), entre muitos outros exemplos.

A importação dos dados significa que os dispositivos em questão estão conectados a uma plataforma IoT, descrita como *"the support software that connects everything in an IoT system"* [7].

A Figura 2.2 promove um exemplo em que diferentes tipos de fontes de dados, tais como dispositivos de medição, automóveis, telemóveis ou hospitais, geram informação que pode ser enviada por diversos meios de conectividade, como o *Wi-Fi*, satélite ou *bluetooth*, até *endpoints*, fins de canais de comunicação, de diferentes protocolos como uma REST API, o Message Queuing Telemetry Transport (MQTT) e o Constrained Application Protocol (CoAP). Nesse ponto, os dados já poderão ser armazenados em sistemas de base de dados ou processados por serviços inerentes à plataforma do processo fazendo com que, dependendo do propósito dos dados, possam ser desenvolvidas aplicações, partindo da partilha desses dados, para fins estatísticos, controlo ou suporte.

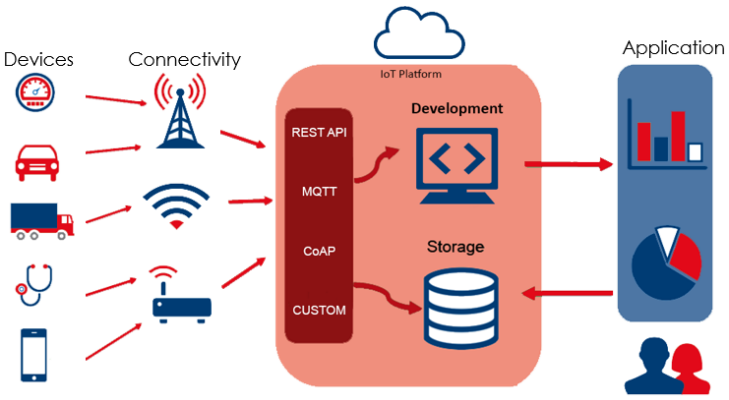

Figura 2.2: Arquitetura de exemplar do funcionamento IoT [8]

2.2 Estandardização em Soluções IoT

Dentro dos sistemas IoT, há um conjunto diversificado e fragmentado de serviços, componentes e aplicações que suportam milhões de objetos IoT a nível mundial, e a tendência será sempre aumentar ano após ano [9]. Nas áreas de IoT, incluindo a de telecomunicações em questão, muitas vezes os sistemas são desenvolvidos sem qualquer estrutura universal coerente.

Com a evolução da indústria, a necessidade para um modelo padrão para a realização de tarefas de *backend*, como processamento, armazenamento e gestão, tem se tornado cada vez mais relevante e é no contexto desse problema que surge este projeto. Com a criação de uma *framework* que siga estes ideais, os sistemas poderão tornar-se modulares ao ponto que poderão ser criadas soluções IoT independentemente da plataforma que se utilize, ou seja, poderá ser utilizado tanto o IoT.nxt [10] mencionado na secção 1.1, assim como a plataforma ThingWorx [11], como outras. Nas próximas secções, estas plataformas e outras irão ser descritas com mais detalhe.

Sucintamente, as plataformas que são utilizadas em soluções IoT permitem conectar *hardware* como, por exemplo, sensores e dispositivos, lidar com diferentes protocolos de comunicação de *hardware* e *software*, fornece segurança e autenticação para dispositivos e para utilizadores, permite armazenar, visualizar e analisar os dados que os sensores e dispositivos captam e permite integrar tudo o que foi descrito acima com sistemas existentes, aplicações e serviços *web* [7].

2.2.1 IoT.nxt

A plataforma que será utilizada para este projeto, e que será essencialmente a fonte dos dados que serão traduzidos, é a IoT.nxt [10] da IoT.nxt, a empresa. Esta plataforma foi previamente selecionada pela Equipa de

IoT da Celfocus [2] sobretudo devido aos recentes projetos que têm sido desenvolvidos usando a plataforma, em parceria com essa mesma empresa, devido à projeção de que poderá haver um domínio nas próximas soluções que poderão surgir sobre o ThingWorx [11]. Sucintamente, as plataformas IoT.nxt [10] e Thingworx [11] são diferentes e com propósitos diferentes. A IoT.nxt traz a componente Edge com o Raptor, tecnologias que serão abordadas em secções posteriores mas que essencialmente permitem a virtualização de projetos mais físicos, enquanto que a Thingworx é mais robusta na parte aplicacional, ou seja, mais indicada para aplicações *web*, interagir com aplicações móveis, entre outras.

Figura 2.3: Logótipo da plataforma IoT.nxt [10]

A IoT.nxt [10] foca-se no fornecimento de soluções inovadoras de *software* e *hardware* para a IoT. Esta empresa desenvolveu uma estrutura de plataforma aberta que é usada para rápido desenvolvimento, integração e implantação de *software* no âmbito da IoT.

Apesar de se ter definido a utilização da IoT.nxt [10], foi realizado um estudo de outras plataformas populares que poderiam também servir para o desenvolvimento do projeto TMForum Standard Translator. Tal como foi supracitado, pretende-se que seja desenvolvido um projeto modular em que facilmente se poderá associar esta *framework* a outras plataformas para o desenvolvimento de novas soluções IoT, de maneira a que não seja obrigatório manter mesma plataforma. As plataformas estudadas serão apresentadas nas subsecções que se seguem.

2.2.2 ThingWorx

A plataforma ThingWorx [11] IoT é particularmente adequada para cenários industriais de IoT. Permite gerir o ciclo de desenvolvimento de aplicações IoT, desde a extração de dados de diferentes tipos de dispositivos através de análises até ao desenvolvimento e implantação IoT. Esta apresenta funcionalidades para que os utilizadores criem aplicações IoT industriais em maior velocidade, sem a necessidade de escrever muitas linhas de código.

Pontos a destacar em relação a esta plataforma são: i) Comunidade de utilizadores; ii) Conectividade. iii) Análise de dados, em que os utilizadores podem extrair dados e realizar análises de IoT diretamente na plataforma ou usar as extensões de análise; iv) A plataforma usa ferramentas prontas e funcionalidades de *drag & drop* para construir soluções na área da Industrial Internet of Things (IIoT); v) Opções de implantação flexíveis, em que os utilizadores podem implantar a plataforma em servidores locais, na nuvem ou em ambientes híbridos.

Com isto, o Thingworx [11] fornece ferramentas e tecnologias que capacitam as empresas a desenvolver e implantar rapidamente soluções IoT complexas. Para contrastar, pode ser indicado que apresenta poucos recursos e suporte de desenvolvimento de código e que as funcionalidades análise de dados e suporte de Inteligência Artificial podem ser consideradas limitadas, mas ainda assim são características que facilmente não são encontradas em outras plataformas.

2.2.3 Record Evolution

O Record Evolution [12] é uma plataforma de IoT que cumpre as funcionalidades padrão desde a recolha de dados de dispositivos IoT, análises avançadas, até ao desenvolvimento e implementação de soluções. É adequada para uma variedade de casos de uso e não está restrito a ape-

nas um setor. Além da recolha de dados e gestão de dispositivos IoT, a plataforma cobre algumas funcionalidades exclusivas, tais como: i) Um estúdio de desenvolvimento de IoT diretamente vinculado a uma loja de aplicações IoT; ii) Desenvolvimento Over-The-Air (OTA); iii) Permite o desenvolvimento de projetos organizados de forma individual, em equipas ou consoante divisões dentro de uma organização especificada; iv) Comunidade de utilizadores.

Apesar de ser uma plataforma completa que permite realizar as tarefas de recolha de dados, análises avançadas, um Integrated Development Environment (IDE) em nuvem para desenvolvimento e implementação de aplicações IoT e gestão de dispositivos, é considerado como um produto relativamente novo, sendo que a fase beta terminou recentemente [13].

2.2.4 Balenda.io

Balena.io [14] é uma plataforma IoT *open source* que oferece um conjunto completo de ferramentas para construir, efetuar *deploy* e gerir conjuntos de dispositivos Linux. É facilmente escalável, o que torna a plataforma adequada para uso em qualquer fase de um projeto de IoT e em uma variedade de casos de utilização que não se limitam a um único setor.

A plataforma pode ser usada para 2 vertentes, de forma independente. Um é a *BalenaCloud*, para realizar o *deploy* de dispositivos e a sua gestão, enquanto que a *OpenBalena* fornece os blocos de construção para que os utilizadores possam criar sua própria forma implantação de dispositivos e a sua plataforma de gestão.

De se estacar em relação a esta plataforma, estão de seguida apresentadas as funcionalidades mais notáveis: i) Colaboração refinada de IoT, de forma a que se pode gerir os privilégios de diferentes utilizadores e definir diferentes tipos de col; ii) Comunidade de utilizadores; iii) Ferramentas de *open source* personalizadas.

Para resumir, esta aplicação tem a seu favor o facto de ser *open source*, ter disponíveis tecnologias envolvendo *clouds* e dispositivos IoT e por ter uma boa base de tecnologia com um longo histórico de presença na comunidade de *open source*. Num aspeto mais negativo, a plataforma é considerada centralizada com forte foco no desenvolvedor e não permite análise de dados ou suporte de Inteligência Artificial, sendo que a última não é relevante para o projeto em questão.

2.2.5 *Particle.io*

Particle.io [15] é um pacote de soluções IoT para *hardware*, conectividade, dispositivos e aplicações. É então considerado como um conjunto de ferramentas de *hardware* e *software* que permite aos seus utilizadores para criar protótipos, dimensionar e gerir diferentes produtos IoT. Como uma plataforma de IoT, o Particle [15] cobre a as funcionalidades IoT necessárias para levar os dispositivos à nuvem, mas também oferece *kits* de desenvolvimento prontos para protótipos, módulos de produção de nível industrial, bem como um sistema de rastreamento e dispositivos configuráveis para o mesmo fim.

As funcionalidades e características mais notáveis desta plataforma são: i) Fácil de usar; ii) Confiabilidade; iii) *All-in-One*, no sentido em que é uma solução completa de *hardware*, *software* e conectividade para que os utilizadores possam investir menos tempo em integrações; iv) Gestão de dados, sendo que quando se trata de tarefas de dados, a plataforma pode ser integrada a qualquer coisa graças à sua API RESTful. v) Gestão de dispositivos e Edge-to-Cloud Stack, em que se combina *hardware*, sistema operacional de dispositivos, conectividade e segurança com diagnóstico de frotas e controlo de acesso para dispositivos.

Sucintamente, esta plataforma é uma oferta completa de dispositivos com conectividade a serviços de plataforma em nuvem com uma boa experiência de utilizador e bom suporte ao desenvolvedor. Para contrastar, é direcionada principalmente para soluções de IoT móveis do tipo *outdoor*,

não permite adaptação para outras plataformas e não apresenta funcionalidades de análise de dados ou suporte de Inteligência Artificial, sendo que a última, mais uma vez, não é relevante para o projeto em questão.

2.2.6 Comparação de Plataformas

Com base nas informações supracitadas sobre cada uma das plataformas e de algumas outras específicas a cada uma delas, foi organizada uma tabela para comparar as funcionalidades e características entre as plataformas, o que permite avaliar de forma objetiva as diferenças entre cada uma delas [13].

Tabela 2.1: Comparação de Plataformas IoT

	Iot.nxt	ThingWorx	Record Evolution	Balenda.io	Particle.io
Integração com Open APIs	✓	✓	✓	✓	✓
Coleção de Dados	✓	✓	✓	✓	✓
Análise Avançada	✓	✓	✓		
Fácil Configuração	✓	✓	✓	✓	✓
Escalabilidade	✓	✓	✓	✓	✓
Colaboração com Comunidade	✓	✓	✓	✓	✓
Criação Simples de Modelos Visuais	✓	✓			✓
Criar Gateways Virtuais	✓	✓		✓	✓

Comparando os resultados das diferentes plataformas obtidos nas categorias de avaliadas e apresentados na Tabela 2.1, a plataforma que se destaca é a da IoT.nxt, sendo possível concluir que oferece soluções IoT dinâmicas, inteligentes e verticais para empresas em todo o mundo pois,

tal como evidenciado anteriormente. A tecnologia exclusiva conecta qualquer dispositivo ou sensor a qualquer aplicativo corporativo, criando uma ponte perfeita entre as fontes de dados, algo que não acontece nas restantes plataformas, pelo menos com a mesma facilidade, sendo o suficiente para se sobrepor à plataforma ThingWorx.

2.2.7 TMForum Open APIs

Open APIs são API acessíveis ao público que foram construídas usando padrões de programação abertos. O TMForum [3] é o órgão da indústria na área das telecomunicações e, em colaboração com seus membros, reuniu um conjunto de API padrão - TMForum Open APIs [16] - para cumprir as funções essenciais de soluções IoT, sendo que o objetivo é que os sistemas sejam integrados com um conjunto padrão de API com uma linguagem comum, então será muito mais simples integrar diferentes sistemas juntos [17].

Apesar de serem muito utilizadas na áreas as telecomunicações, o alcance, a nível de áreas, destas API é amplo.

As TMForum APIs são baseadas na arquitetura Representational State Transfer (REST), o que faz com que forneçam uma maneira flexível e leve de integrar aplicações e surgiram como o método mais comum para conectar componentes em arquiteturas de microsserviços.

As 3 principais API que serão utilizadas como "linguagem destino" da tradução que irá ser feita serão o Resource Inventory Manager API, o Alarm Management API e o Geographic Site API.

Resource Inventory Manager API

O Resource Inventory Manager API [1] permite o armazenamento e gestão de instâncias de recursos com dados para diversas situações, tornando-se assim num inventário tal como o nome sugere. Com isto, esta API permite listar os recursos existentes que poderão ser utilizados para associar a informação que consta nas restantes API.

Alarm Management API

O Alarm Management API [4] fornece a interface de cliente padronizada para sistemas de gestão de alarmes para criar, monitorizar e gerir alarmes. A interface oferece tanto funcionalidades de gestão de alarmes em recursos como de serviços. Os objetos alarmado poderão ser objetos armazenados pela Resource Inventory Manager API mas não estão restritos a nenhuma tecnologia ou fornecedor específico, portanto, a API pode ser usada em uma ampla variedade de casos de gestão de falhas.

Assim, sucintamente, esta API tem como propósito desde permitir que seja feita a subscrição de funções de gestão, ou seja a realização de notificações de alarme ”simples”, até à sincronização de sistemas de gestão em situações de em eventos de alarme, alarmes de ultrapassagem de limiares, *acknowledges*, entre outras.

Geographic Site API

Geographic Site [5] é uma especialização da entidade abstrata Place, assim como Geographic Address [18] e Geographic Location [19], as quais têm igualmente uma API própria e específica, sendo que em conjunto estão organizadas tal como apresentado na Figura 2.4.

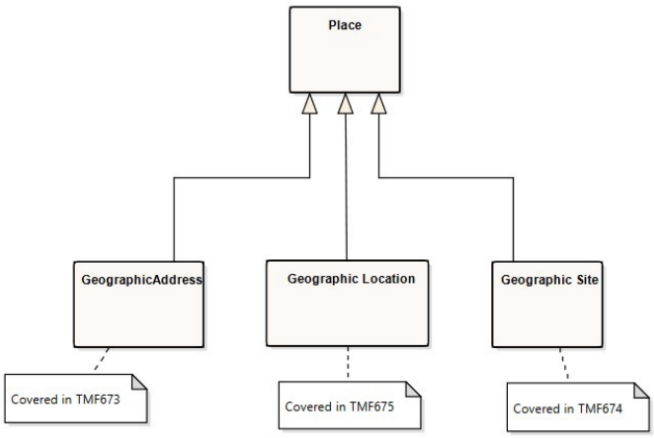

Figura 2.4: Modelo da entidade Place [5]

A Geographic Site API, só por si, permite criar, atualizar e obter *geographic sites*, que essencialmente tratam-se de um conjunto de dados que representam uma localização. A API apresenta as operações para gerir *sites* (criar, ler, excluir) que podem ser associados a um cliente, uma conta, a um serviço de entregas ou outras entidades. Esta API define um *site* como uma classe de conveniência que permite facilmente referir-se a lugares importantes para outras entidades, onde um lugar geográfico é a entidade que pode responder à pergunta “onde?”, permitindo determinar onde as coisas estão em relação à superfície terrestre.

Resumidamente, esta API permite: i) Criar um novo *site* ou remover um existente; ii) Atualização parcial de um site geográfico; iii) Obter uma lista de sites armazenados em um servidor filtrado por um determinado critério; iv) Obter um site específico; v) Notificação de eventos em um site geográfico.

Soluções Semelhantes

Existem API com funcionalidades semelhantes às que foram apresentadas e que poderiam ser consideradas como alternativas viáveis. Um exemplo disso é a Oracle Resource Management API [20] que, assim como a Resource Inventory Manager API, foi projetada como um meio de gerir recursos, mas que neste caso é capaz de gerir utilizadores por meio da interface SOAP. Apesar disso, qualquer TMForum API sobrepõe-se neste contexto, visto que o TMForum [3] disponibiliza um conjunto de API padrão para esta área e, tal como referenciado inicialmente na secção 2.2.7, por ser usada uma linguagem padrão, é muito mais simples o processo de integração com novos sistemas e API.

Capítulo 3

Análise e Arquiteturas

De forma a conseguir desenvolver um modelo capaz de satisfazer os requisitos do projeto, é necessário que estes sejam definidos e analisados previamente. Sendo assim, segue-se uma análise dos requisitos funcionais e não-funcionais do trabalho desenvolvido, secção 3.1 juntamente com os casos de utilização, secção 3.2.

3.1 Requisitos

Os requisitos são condições que a aplicação deve preencher, sendo estes funcionais, subsecção 3.1.1, ou não funcionais, subsecção 3.1.2. São categorizados como "Evidente" todos os requisitos que, quando executados, o utilizador está ciente do evento, e como "Invisível" quando o requisito está a ser realizado sem que o utilizador esteja inteirado desse mesmo evento.

3.1.1 Requisitos Funcionais

Requisitos funcionais traduzem eventos, da perspetiva do utilizador, que o sistema faz, apresentados na Tabela 3.1, onde a coluna "Descrição" indica a função do requisito e a coluna "Categoria" indica o tipo de requisito em questão conforme as categorias supracitadas.

Tabela 3.1: Requisitos funcionais

Ref.#	Descrição	Categoria
RF1	Tradução de Alarmes IoT.nxt-TMForum	Evidente
RF2	Gestão de Alarmes	Evidente
RF3	Tradução de Recursos IoT.nxt-TMForum	Evidente
RF4	Gestão de Recursos	Evidente
RF6	Tradução de Informações Geográficas IoT.nxt-TMForum	Evidente
RF5	Gestão de Informações Geográficas	Evidente

Pretende-se então que o projeto permita realizar o processo de tradução de dados para as API do TMForum de alarmes, recursos e informações geográficas. Ao fazê-lo, os dados traduzidos serão armazenados num sistema de base de dados onde será possível geri-los através de pesquisas, filtrações, atualizações e exportações, por exemplo.

3.1.2 Requisitos Não Funcionais

Uma vez que os requisitos funcionais indicam o que o sistema faz, os requisitos não funcionais indicam como o sistema os faz, geralmente sendo ocultos ao utilizador. Estes tipos de requisitos estão apresentados na Tabela 3.2, onde, à semelhança da Tabela 3.1, a coluna "Descrição" indica a função do requisito e a coluna "Categoria" o tipo de requisito.

Tabela 3.2: Requisitos não funcionais

Ref.#	Descrição	Categoria
RNF1	O sistema deverá ser simples e intuitivo	Evidente
RNF2	Os micro-serviços do sistema deverão suportados por Node.js	Invisível
RNF3	Os dados dos micro-serviços deverão ser armazenados em MongoDB	Evidente
RNF4	O sistema deverá ser modular para ser adaptado a diferentes plataformas IoT	Evidente
RNF5	O sistema deverá realizar uma tradução de informação entre API IoT.nxt e API TMForum IoT	Evidente
RNF6	O sistema deverá expor em TMForum uma lista de alarmes IoT.nxt atualizados por demanda	Evidente
RNF7	O acesso aos dados das plataformas deverá depender de um método de autenticação	Evidente

Os requisitos apresentados encontram-se nas categorias de usabilidade e eficiência, através dos requisitos RNF1, RNF3, RNF5 e RNF6, portabilidade, através do requisito RNF4, e confiabilidade, através do requisito RNF7, para proporcionar uma experiência agradável ao utilizador.

3.2 Casos de Utilização

Casos de utilização são narrativas que descrevem uma sequência de possíveis eventos entre dois atores na utilização de um sistema. Sendo assim, os casos de utilização da aplicação são a realização das traduções dos diversos pedidos e a gestão dos dados traduzidos, ilustrados na Figura 3.1.

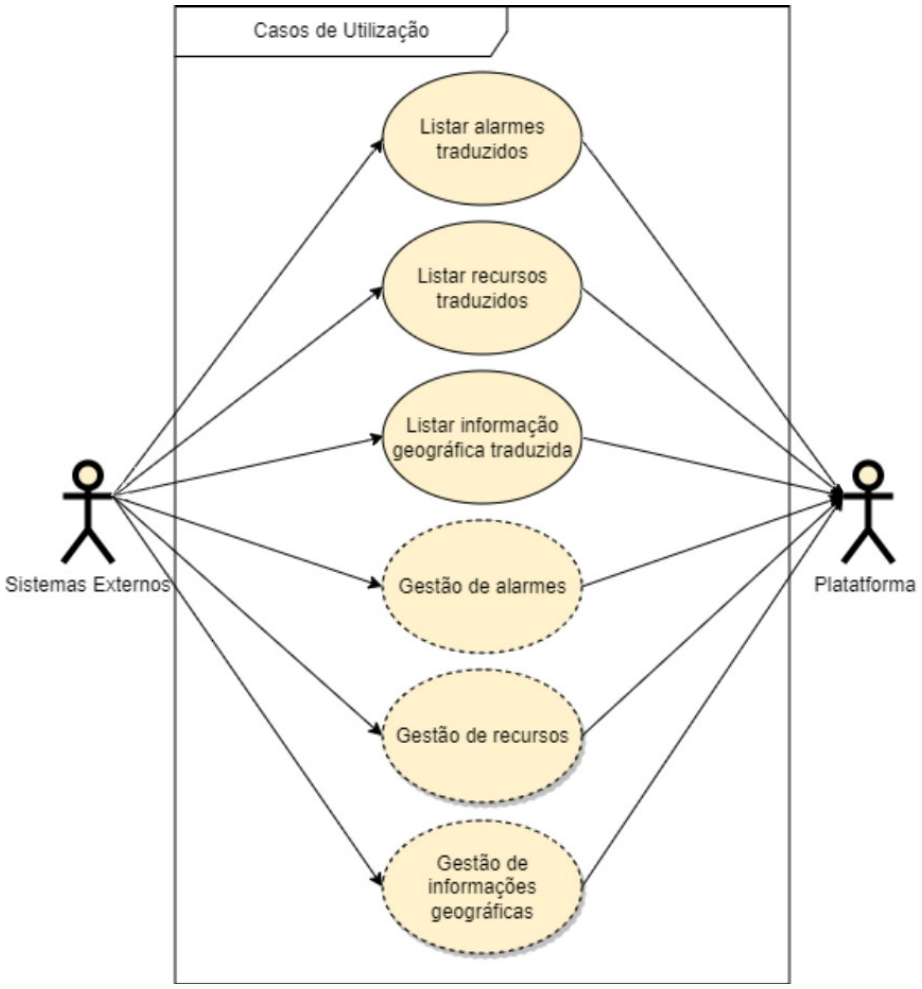

Figura 3.1: Casos de utilização dos sistemas recorrendo ao TMForum Standard Translator

Como podemos observar na Figura 3.1, existem 3 casos de utilização principais, os quais permitem, aceder ao conteúdo das API que estão a ser utilizadas na plataforma IoT, listando-o do lado do sistemas externos com linguagem TMForum (Listar alarmes, recursos e informação geográfica traduzida). Adicionalmente foram descritos outros 3 casos de utilização referentes à gestão do conteúdo das 3 API da plataforma IoT, que se tratam essencialmente das operações de criação, atualização e eliminação de objetos, mas como não são o foco deste projeto estão marcadas a tracejado (Gestão de alarmes, recursos e informações geográficas). Assim sendo, com esta fase do projeto definida, foi possível projetar a arquitetura do sistema que permite a ligação entre as API das plataformas IoT com os sistemas externos, tal como se pode verificar a Figura 3.2.

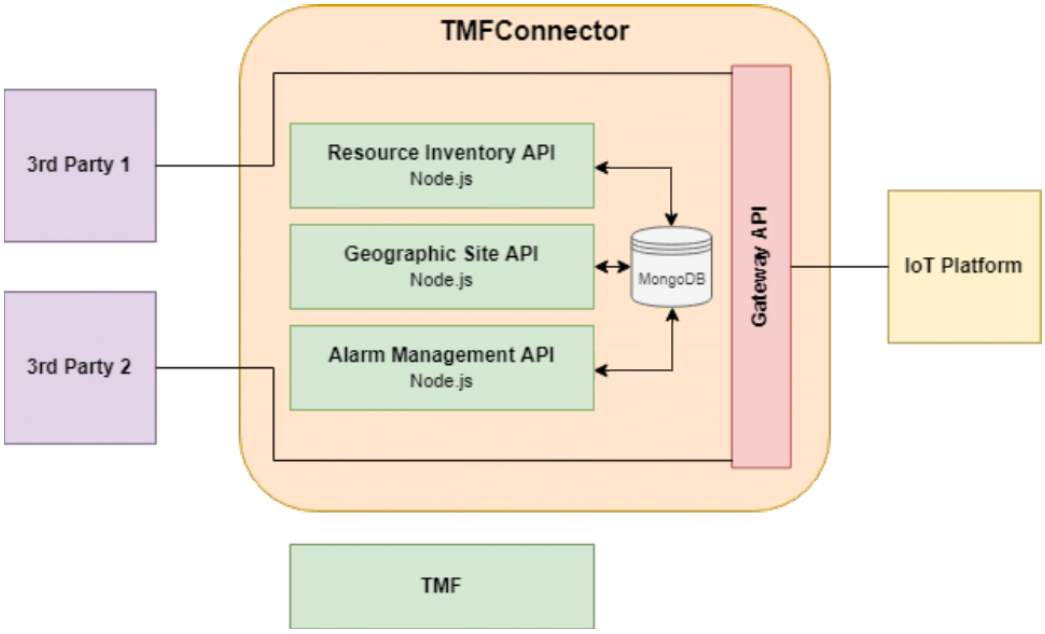

Figura 3.2: Modelo abstrato do TMForum Standard Translator

Segundo o que foi projetado, a Gateway API comunica com a plataforma IoT para a obtenção dos dados e é nessa API que os processos de tradução dos dados são realizados e validados por cada uma das API do TMForum e armazenados em bases de dados da MongoDB. Os dados poderão ser obtidos por sistemas externos por comunicação com a Gateway API que permite aceder, através de si, aos dados traduzidos por meio de

routing, que é essencialmente o encaminhamento dos dados para *end-points* do *gateway*, tornando-se assim o intermediário entre os pedidos de sistemas terceiros e as API do TMForum.

3.3 Arquitetura Adicional

Sendo também um dos objetivos do estágio na Celfocus, e inserindo-se também no projeto ao qual pertence o TMForum Standard Translator, foi necessário paralelamente o desenvolvimento de uma *framework* capaz de criar e simular um *site* digitalmente com o intuito dos dados gerados por esses *sites* serem usados em testes, tais como os necessários para o projeto em questão.

Um *site* é essencialmente uma estrutura que compreende um equipamento de rede elétrica e a infraestrutura de dispositivos associados. Esta infraestrutura pode variar desde a associação de diferentes tipos de dispositivos, tais como *AC Meters*, *DC Meters*, *Rectifiers*, *Modem Managers*, alarmes e Raptors [21]. Os Raptors [21] são um tipo de *gateways* da IoT.nxt equipados com a capacidade de se ligarem à maioria dos sistemas do ecossistema IoT, podendo assumir as propriedades e protocolos necessários para comunicar com qualquer dispositivo ou sensor [21], o que permite captar os seus dados e encaminhá-los para a plataforma IoT em questão.

Esta ferramenta é considerada como uma mais valia pois, no contexto da instalação de *sites* reais, existe um longo processo entre planeamentos, agendamentos favoráveis entre as entidades envolvidas, deslocações, custos, montagens e manutenções, processos que são evitados quando se tratam de *sites* criados digitalmente, sobretudo quando são necessários apenas para fins de teste.

Para possibilitar o desenvolvimento desta *framework* que serviu de apoio ao TMForum Standard Translator, fez-se uso do Edge Raptor [22],

que é uma *gateway* da IoT.nxt tal como o dispositivo Raptor [21] supracitado, mas que se trata de um *software* abrangente pois permite uma implantação rápida e é intuitivo o suficiente para permitir a integração com outros sistemas.

Sucintamente, este Edge Raptor é instalado numa máquina virtual configurada com o Ubuntu [23] por meio do *package* raptorsnap [24]. Com o Edge Raptor instalado e devidamente configurado, é possível dar uso à sua interface para criar um servidor MQTT. O MQTT trata-se de um protocolo de mensagens simples que permite distribuir informações de telemetria entre um servidor e os clientes a ele ligados. O servidor criado irá servir como um *site* virtual, pois nessas condições é possível criar diversos tópicos, que são os destinos das mensagens trocadas e que representarão cada dispositivo do *site*, e finalmente cada tópico terá um conjunto de propriedades que representam as propriedades do dispositivo em questão. Para tratar da simulação e fazer o servidor MQTT funcionar como um *site* real, criou-se um microsserviço em Python que se liga ao servidor MQTT criado e, para cada propriedade de cada dispositivo, representados por tópicos, são gerados valores usando um *algoritmo* que permite personalizar o tipo de variação, os extremos de variação de uma iteração para outra e os extremos do valor de uma forma geral. Sempre que os valores são calculados, são enviados para a propriedade indicada do tópico em questão usando mensagens do tipo *publish*.

O diagrama apresentado na Figura 3.3 mostra a arquitetura utilizada para a simulação de *sites* virtuais.

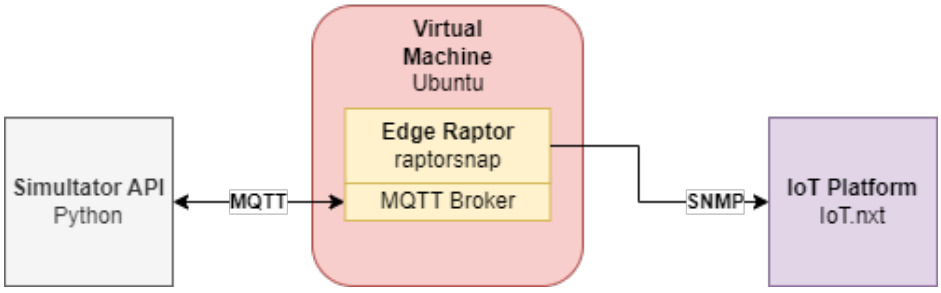

Figura 3.3: Arquitetura de Simulação de *sites* virtuais

Finalmente, o *Edge Raptor* ao ser devidamente configurado, envia os dados recebidos pelo microsserviço criado em Python e encaminha-os para a plataforma *IoT.nxt* para poderem ser utilizados em testes do TM-Forum Standard Translator.

Capítulo 4

Implementação do Modelo

Neste capítulo iremos descrever a forma como foram implementadas cada uma das fases do projeto, tal como será feita uma análise das melhores escolhas entre as possíveis opções de implementação. Na secção 4.1 será descrita a forma como foi desenvolvido o *API Gateway* necessário para a comunicação entre sistemas e onde será feita a transformação dos dados recebidos pela plataforma IoT e na secção 4.2 será apresentado a forma como foram criados os microsserviços que suportam as API TMForum.

4.1 API *Gateway*

Tendo em conta o modelo de arquitetura representado na Figura 3.2, ficou definido que seria necessária a criação de uma entidade que sirva como intermediária entre o conteúdo das API das plataformas IoT e com os microsserviços com as API TMForum. Essencialmente, a sua função será obter o conjunto dos objetos armazenados pelas API implementa-

das que se pretende traduzir, realizar a tradução e encaminhar os dados resultantes para os microsserviços correspondentes. Para isso, a plataforma escolhida para esta tarefa foi o Spring Boot [25]. O *Java Spring Framework* é uma estrutura de nível empresarial popular e *open source* utilizada para criar aplicações independentes que são executadas na *Java Virtual Machine* (*JVM*). Esta estrutura é utilizada pelo *Java Spring Boot*, sendo este uma ferramenta que torna o desenvolvimento de aplicações web, *gateways* e microsserviços com *Spring Framework* mais simplificado. Contudo, irá ser focada essencialmente a vertente do *gateway* pois, utilizando o Spring Boot [25], passa-se a ter uma maneira simples e eficaz de rotear solicitações recebidas para o destino apropriado usando o Mapeamento do *Gateway Handler*. A arquitetura que suporta o funcionamento do *gateway* encontra-se apresentada na Figura 4.1.

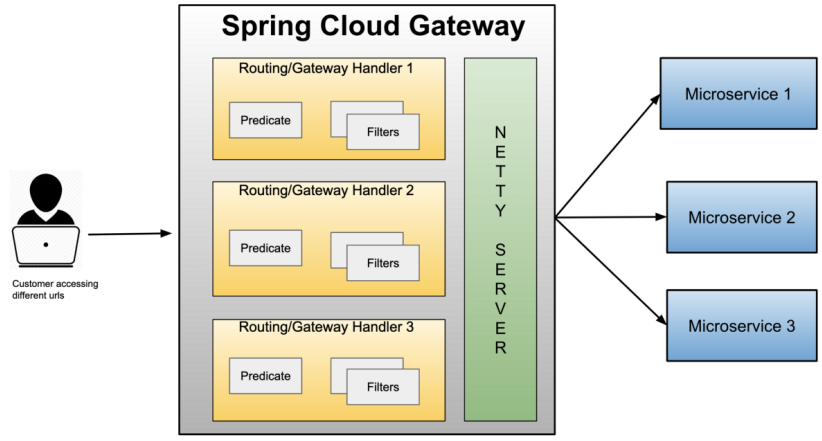

Figura 4.1: Arquitetura *Spring Cloud Gateway* [26]

O *Spring Cloud Gateway* consiste em 3 blocos de construção principais. O *Route* é considerado o destino para o qual se quer que um pedido específico seja roteado. É composto por um Uniform Resource Identifier (URI) de destino, uma sequência de caracteres que identifica um recurso, uma condição que deve satisfazer, mais conhecida por *Predicate* e um ou mais filtros definidos no atributo *Filter*. O predicado é uma condição que deve ser correspondida, ou seja, caso os pedidos tenham algo como um *path* em específico ou certo cabeçalho. Já os filtros são instâncias de *Spring Framework WebFilter*. É através deste atributo que é possível

modificar o pedido ou a resposta. A personalização do tipo de encaminhamento que será feito é através deste atributo de forma a cumprir com as necessidades do utilizador. [26]

Para a sua implementação foi necessário recorrer ao Spring Tools 4 [27], uma ferramenta que pode ser encontrado na *marketplace* do ***Eclipse*** [28], o *IDE* utilizado neste projeto, a qual permite a fácil criação de projetos *Spring* com dependências escolhidas pelo utilizador, conforme as suas necessidades. Sendo que a intenção é de se montar um *gateway*, as dependências selecionadas foram o *Spring Boot DevTools*, permitindo reinicializações rápidas de aplicações, *LiveReload* e configurações para uma experiência de desenvolvimento aprimorada e o *Gateway*, fornecendo uma maneira simples, mas eficaz, de fazer *routing* para API. O resultado desta operação é a geração da classe apresentada no Código 1, a partir da qual é inicializada a aplicação, assim como a geração do ficheiro application.yaml, posteriormente modificado para application.properties pois torna-se mais fácil e intuitiva a declaração das rotas, onde é possível declarar todas operações de *routing* necessárias seguindo a lógica dos predicados e dos filtros apresentada na Figura 4.1 e que já foi mais detalhada anteriormente.

Código 1: Base da APIGateway

```
@SpringBootApplication
public class GatewayApiApplication {
  public static void main(String[] args) {
    SpringApplication.run(GatewayApiApplication.class, args); }}
```

4.2 Microsserviços

Microsserviços são um tipo inovador de arquitetura de software, que consiste em construir aplicações desmembrando-as em serviços independentes. Como uma *framework* de arquitetura, microsserviços são distribuídos e levemente acoplados. Estes serviços normalmente um conjunto próprio de tecnologia, o que se refere ao modelo de base de dados e de gestão de dados, e comunicam entre si recorrendo a API. Tal como a Figura 3.2 sugere, esse facto pode ser verificado na mesma em que a cada API está associada uma coleção numa base de dados implementada através de MongoDB [29].

Na página da TMForum [3], existe uma tabela com o conjunto de todas as *Open API* disponíveis, cada uma associada com uma amostra de código que foi utilizado para finalizar a sua implementação e que pôde ser executado recorrendo ao *software* Node.js [30]. Com as API em execução, o *software* que serviu como auxílio para o teste de cada uma delas foi o Postman [31]. O Postman [31] é uma plataforma de API para construir e usar diferentes API. Neste contexto foi utilizada precisamente para testar a criação dos primeiros objetos de cada uma das API antes de serem utilizadas para o armazenamento dos objetos traduzidos provenientes da plataforma IoT.nxt [10].

4.3 Microsserviço - Alarm Management API

Nesta fase do projeto, foi feita uma análise à estrutura de dados da Alarm Management para projetar a maneira como seria possível mapear os dados dos alarmes provenientes da plataforma IoT.nxt, foram definidas as regras de mapeamento e finalizou-se com a implementação da obtenção dos alarmes e da tradução dos mesmos.

4.3.1 Análise da estrutura dos dados

Para a utilização desta API, existe uma estrutura bem definida que indica a obrigatoriedade e não obrigatoriedade da definição de certos atributos, quando os alarmes são criados, que pode ser encontrada na documentação [4] da mesma, mas que essencialmente exige que o conjunto chave-valor dos campos *alarmType*, *perceivedSeverity*, *probableCause*, *alarmedObject*, *sourceSystemId*, *state* e *alarmRaisedTime* sejam definidos pois são considerados como essenciais. Os restantes atributos são considerados como opcionais mas, no contexto deste projeto, em que se pretende realizar a tradução da informação de alarmes de IoT.nxt [10] para TMForum [3], poderá ser obrigatória a sua definição no sentido de não perder informação que já se encontre previamente definida, ou seja, caso um atributo já exista no lado da plataforma IoT, e mesmo que o seu equivalente seja opcional, pretende-se que passe a ser essencial a sua definição do lado do TMForum [3] para não causar perda de informação. O conjunto com a identificação de todos os atributos dos alarmes existentes pode ser verificado na Tabela 4.2, tal tabela que foi construída com base na documentação [4] da API em questão.

Tabela 4.1: Atributos dos Alarmes TMForum [1]

Atributos dos Alarmes - Alarm Management API TMForum				
Obrigatórios	**Opcionais**			
alarmType	@baseType	alarmChangedTime	clearUserId	place
perceivedSeverity	@schemaLocation	alarmClearedTime	comment	plannedOutageIndicator
probableCause	@type	alarmDetails	correlatedAlarm	proposedRepairedActions
alarmedObject	ackState	alarmEscalation	crossedThresholdInformation	reportingSystemId
sourceSystemId	ackSystemId	alarmReportingTime	externalAlarmId	serviceAffecting
state	ackUserId	alarmedObjectType	isRootCause	specificProblem
alarmRaisedTime	affectedService	clearSystemId	parentAlarm	

4.3.2 Teste de publicação de alarmes

Numa fase inicial, testou-se a publicação de alguns alarmes com informação previamente definida em formato JavaScript Object Notation (JSON) através do Postman [31]. Ao serem publicados na API através do método POST, os mesmos ficam armazenados numa base de dados MongoDB [29].

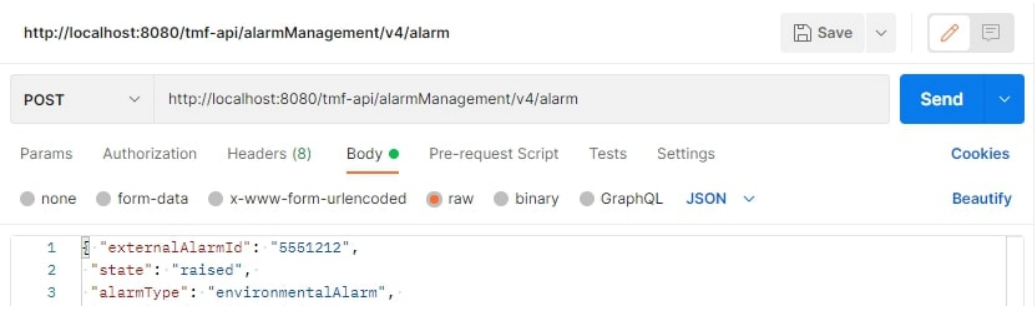

Figura 4.2: Excerto do pedido de publicação de um alarme na *Alarm Management* API *TMForum* [4]

A Figura 4.2 mostra um excerto de um exemplo de teste de publicação de alarme usando o método POST da API de alarmes do TMForum e, como se pode constatar, é utilizado o formato JSON com a definição

dos campos apresentados na Tabela 4.2, especificamente os obrigatórios alguns dos restantes. A Figura 4.3 permite verificar o resultado desta publicação.

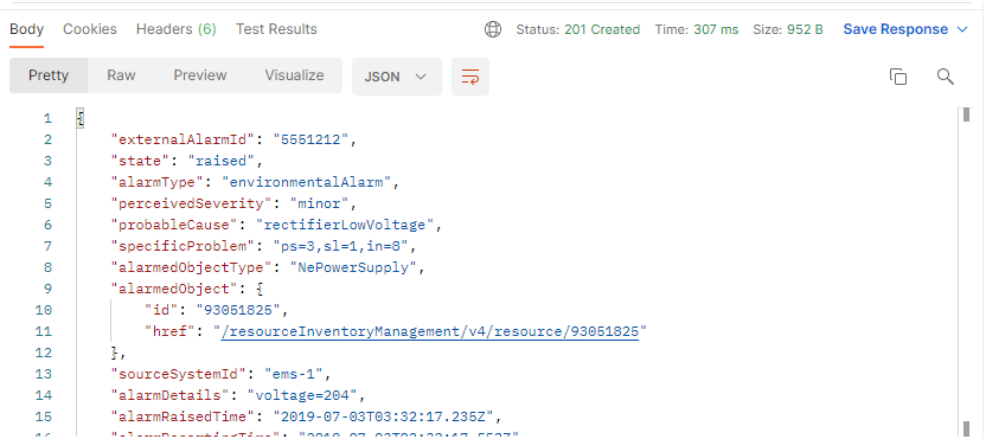

Figura 4.3: Corpo de resposta pedido de publicação de um alarme na Alarm Management API TMForum [4]

Como evidenciado na Figura 4.3, confirma-se o sucesso da publicação através do status http "201 Created", o qual é utilizado como resposta indicando que a requisição foi bem sucedida e que um novo recurso foi criado, que neste caso se trata de um alarme. Comprova-se também que o pedido apresentado na Figura 4.2 corresponde a esse resultado pois a resposta indica o sucesso mostrando que, por exemplo, o campo *externalAlarmId* tem o mesmo valor de *5551212* que foi indicado no pedido, ou seja, o sucesso também se comprova pela confirmação da definição requisitada para cada atributo através da sua apresentação na resposta. Com a conclusão desta fase teste passa a ser possível a criação de novo alarmes após serem traduzidos para objetos compatíveis com a estrutura da TMForum.

Finalizou-se este teste tentado obter os alarmes criados através do *gateway* implementado, que acede à informação existente na API da TMForum [3] que está em execução no endereço *localhost:8080/tmf-api/alarmManagement/v4/alarm*, reencaminhando para uma rota no seu próprio en-

dereço. A criação deste encaminhamento encontra-se declarada no Código 2.

Código 2: Defination de uma rota no *gateway*

```
server.port=9005
spring.cloud.gateway.routes[0].id= alarm-service
spring.cloud.gateway.routes[0].uri= http://localhost:8080
spring.cloud.gateway.routes[0].predicates= Path=/alarmManagement/**
spring.cloud.gateway.routes[0].filters[0]=RewritePath= /alarmManagement,/tmf-api/
    alarmManagement/v4
```

Com isto, através do *gateway*, o conteúdo da API da TMForum [3] é roteada para o endereço *localhost:9005/alarmManagement/alarm* e obtém-se o resultado apresentado na Figura 4.4.

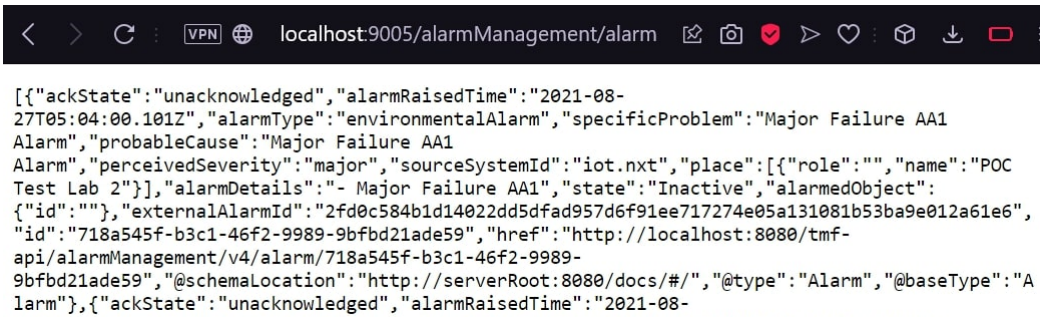

Figura 4.4: Excerto do resultado do roteamento da informação da API *TMForum*

Tal como se pode constatar, apenas foi apresentado um excerto que permite percecionar o sucesso do processo de roteamento e a obtenção dos alarmes traduzidos, em formato JSON através do *gateway* criado, seguindo a regra de roteamento defindo no Código 2.

4.3.3 Definição das regras de mapeamento

Para se poder realizar a tradução dos alarmes, foi necessário estudar a estrutura dos alarmes da plataforma IoT.nxt [10] comparando-a com a utilizada na TMForum [3], de forma a concluir o tipo de semelhanças e de diferenças existentes. Nestas condições, foi possível desenhar um esquema que basicamente indica o mapeamento que será necessário realizar a partir dos alarmes da plataforma IoT tanto a nível de campos como a nível de dados para que, seja possível obter um JSON válido para ser publicado na API dos alarmes da TMForum [3]. O método utilizado foi essencialmente associar os campos de ambas as API que armazenam o mesmo tipo de informação como um par de mapeamento, assim como aqueles campos que apesar de não significarem exatamente a mesma coisa, são considerados os campos mais apropriados para armazenarem esse tipo de informação.

No caso do mapeamento de dados, por exemplo, o campo *priority* do IoT.nxt [10] pode ser definido com os valor *High* e *Low*, e o mesma corresponde, ao campo *perceivedSeverity* cujos valores são consistentes com a *ITU-T Recommendation X.733* [32], um documento que especifica um conjunto de recomendações para a especificação destes alarmes, que obriga a que o seu valor seja um entre *cleared*, *inderteminate*, *critical*, *major*, *minor* e *warning*, ou seja, nesta situação, mapear a informação de um campo para o outro não seria suficiente pois os valores *High* e *Low* não se encontram entre as opções possíveis. Para resolver essa situação, nesses casos é necessário mapear o valor dos campos e no exemplo apresentado anterior, o *High* corresponderia ao *major* e o *Low* ao *minor*. Assim, foi desenhado um diagrama que permite uma visualização dos vários mapeamentos que serão realizados, tanto a nível de campos como de dados, e que pode ser verificado na Figura 4.5.

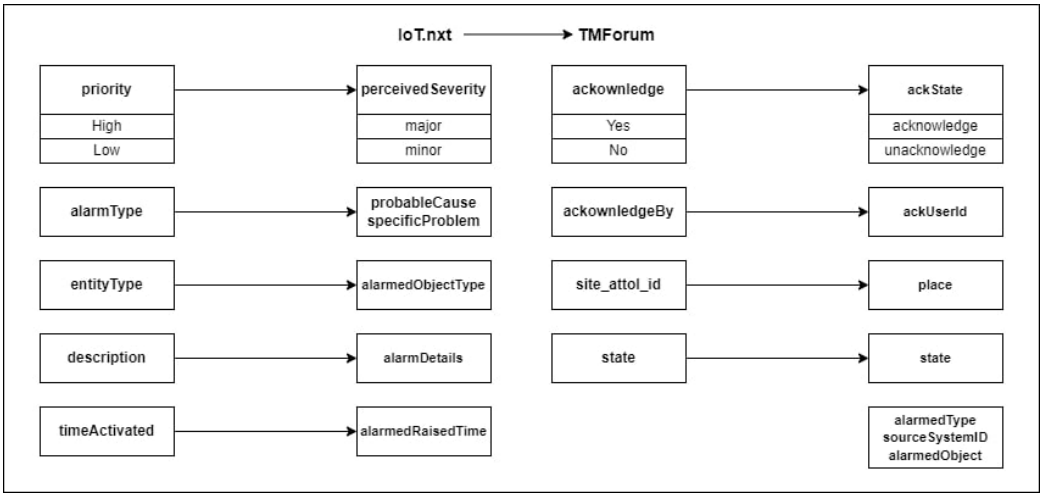

Figura 4.5: Planificação das regras de mapeamento de campos e de dados

No final do diagrama encontram-se alguns campos que não têm qualquer correspondência os campos da IoT.nxt [10]. Isto deve-se pois tratam-se de campos que são obrigatórios quando os alarmes do TMForum [3] são criados, por isso, devem ser atribuídos valores padrão, como por exemplo ao *sourceSystemID* em que se sabe que a origem dos alarmes é a plataforma IoT em questão, por isso foi-lhe atribuído o valor de "*IoT.nxt*", ou simplesmente ficam com uma definição vazia, como por exemplo o *alarmedObject* na situação em que não há nenhum objeto alarmado definido.

4.3.4 Tradução de alarmes de IoT.nxt para TMForum

Nestas condições foi possível iniciar o processo de implementação das regras de mapeamento que foram apresentadas. No projeto do *gateway* criou-se uma classe utilitária para albergar todos os métodos necessários para realizar este processo, entre eles o acesso à fonte dos alarmes da IoT.nxt [10], a tradução propriamente dita e a publicação do resultado da tradução na API TMForum. Em fase inicial, foi fornecida uma coleção de amostra com alguns alarmes provenientes da IoT.nxt [10] em formato JSON *Array* e, para simular o acesso à plataforma IoT, foi criado um mi-

crosserviço à parte utilizando o Spring Boot que foi populado com essa mesma coleção replicando o comportamento de uma fonte da plataforma IoT. A coleção em questão armazena os alarmes num ficheiro de texto, o qual é acedido pelo microsserviço, apresentando o seu conteúdo sempre que houverem pedidos Hypertext Transfer Protocol (HTTP) GET para */alarms*. O código utilizado para este fim encontra-se presente no Código 3.

Código 3: Função para a criação do microsserviço de teste

```
@GetMapping("/alarms")
  public String all() throws IOException {
    String file ="alarms.txt";
    File resource = new ClassPathResource(file).getFile();
     String text = new String(Files.readAllBytes(resource.toPath()));
     JSONObject j = new JSONObject(text);
    return JSONObject.valueToString(j.get("result")); }
```

Após a criação deste microsserviço, quando em execução, é possível acedê-lo dentro do controlador do *gateway*, onde será feito todo o processo de tradução, através do Código 4.

Código 4: Função para a obtenção dos alarmes do microsserviço de teste

```
public String getOrPostTest() {
    String GET_URL = "http://localhost:3344/alarms";
    RestTemplate restTemplate = new RestTemplate();
    Map<String, String> params = new HashMap<String, String>();
    String result = restTemplate.getForObject(GET_URL, String.class, params);
    return result;}
```

Agora sendo possível aceder a alarmes da IoT.nxt [10], foi criado um método que os traduza e que os publique na API TMForum. A primeira tentativa de um método de tradução foi realizar um conjunto de declarações *if* que se certificavam da existência de certos campo predefinidos

e, caso essas condicões se verificassem, eram criados adicionados campos a um JSON novo com base nas regras de mapeamento declaradas anteriormente. Um excerto deste método pode ser verificado no Código 5.

Código 5: Função de tradução de alarmes IoT.nxt

```
public static JSONObject iotnxtToTmforum(JSONObject iotJson) {
  JSONObject tmfJson = new JSONObject();
  iotJson.keySet().forEach(key ->{
  Object keyvalue = iotJson.get(key);
  String value = (String) keyvalue;
    tmfJson.put("alarmType", "environmentalAlarm");
    tmfJson.put("sourceSystemId", "iot.nxt");
    JSONObject alarmedObject = new JSONObject();
    alarmedObject.put("id", "");
    tmfJson.put("alarmedObject", alarmedObject);
  if(key.equals("priority")) {
      String priority =  iotJson.get(key).toString();
      String perceivedSeverity ="";
      if(priority.equalsIgnoreCase("High"))
      perceivedSeverity = "major";
      else if(priority.equalsIgnoreCase("Low"))
        perceivedSeverity = "minor";
      tmfJson.put("perceivedSeverity",perceivedSeverity); }
  ...
  else if(key.equals("state")) {
      tmfJson.put("state", iotJson.get(key));}});
  return tmfJson;}
```

A tradução pôde ser feita com sucesso mas, apesar de se tratar de um excerto e com base nas regras que foram apresentadas na Figura

4.5, é percetível que o código completo acaba por ser exaustivo devido à quantidade de condições precisas e torna-se pouco prático que, em caso de necessidade de adição de novas regras a este modelo, que se tenha de fazer novamente alterações no código, o que impossibilita que haja estabilidade neste processo.

Após alguma pesquisa, foi possível perceber que a utilização da base de dados seria uma boa aposta como alternativa no sentido em que se torna mais prático fazer alterações/adições nas regras de mapeamento e o código necessário é mais estável e genérico ao ponto poder ser utilizado para traduções de IoT.nxt [10] para TMForum [3], vice-versa ou entre diferentes plataformas IoT, desde que a estrutura das regras na base de dados seja mantida.

Para esta tarefa manteve-se a mesma base de dados referida anteriormente, a MongoDB [29], tendo-se criada uma coleção denominada de *Mapping Type* que poderá no futuro vir a guardar diferentes tipos de mapeamento de alarmes, tal como já foi mencionado. A estrutura utilizada pode ser verificada na Figura 4.6 em que o campo *mappingType* identifica o tipo de mapeamento, as chaves de nível 1 (*priority*, *alarmType*, *entityType*, etc.) representam os campos IoT.nxt [10] que serão traduzidos, dentro deles o valor do campo *property* é o correspondente em TMForum [3] e, finalmente, tenham o campo *value*, o seu valor engloba todas as possibilidades de mapeamento de dados.

Figura 4.6: Definição das regras de mapeamento de alarmes de campos e de dados em MongoDB

Estando neste ponto, apenas falta aceder ao conjunto de regras que foram criadas na base dados e aplicá-las aos alarmes para teste que foram adicionados ao microsserviço que foi criado anteriormente. Para isso produziu-se a função *mapAlarms()* presente no Código 6 que, por cada alarme que se pretende traduzir, irá ser feita uma iteração por todas as chaves existentes. Caso as chaves tenham correspondência com alguma chuva existente nas regras de mapeamento que foram obtidas também na função, irá ser adicionado a um novo objeto JSON a nova chave com o mesmo valor, e só quando o campo *value* existe é que será necessária a realização do mapeamento do valor do campo, sendo nesse caso criado tanto uma nova chave como um novo valor, que serão também adicionados ao novo alarme traduzido.

Código 6: Função de mapeamento de alarmes com regras na base de dados

```
static JSONObject mapAlarms(JSONObject iotJson, String mappingType) {
    Document rules = getAlarmMappingRules(mappingType);
    JSONObject tmfJson = new JSONObject();
     iotJson.keySet().forEach(key ->{
```

```
        Document propertyDoc =((Document)rules.get(key));
        if(propertyDoc!=null) {
        propertyDoc.keySet().forEach(pkey->{
          if(pkey.contains("property")) {
            String newKey = propertyDoc.get(pkey).toString();
            String newValue = iotJson.getString(key);
            if(((Document)rules.get(key)).get("value")!=null) {
              newValue = ((Document)propertyDoc.get("value")).getString(newValue)
    ;}
            tmfJson.put(newKey, newValue);}});});
     Document defaultProperties = ((Document)rules.get("default"));
     defaultProperties.keySet().forEach(key->{
        String dValue=""; JSONObject json;
        String typeName =  defaultProperties.get(key).getClass().getSimpleName();
        if(typeName.equals("String")) {
         dValue = defaultProperties.get(key).toString();
          tmfJson.put(key, dValue);}
        else {
          dValue = ((Document) defaultProperties.get(key)).toJson();
          json = new JSONObject(dValue);
          tmfJson.put(key, json);}});
    return tmfJson;}
```

Tal como referido anteriormente, a estrutura das regras de mapeamento existentes na base de dados foi criada de tal forma para que a função *mapAlarms()* pudesse manter-se inalterada mesmo que se criem novos tipos de mapeamento no futuro, sendo então o mais genérica possível.

Finalmente, com acesso a alarmes da IoT.nxt [10] e com a função de mapeamento criada, criou-se uma última função no *gateway* que acedendo aos alarmes da plataforma IoT, realiza a tradução de cada um deles

agrupando-os num JSONArray e publicando-se na TMForum Alarm Management API [4]. A implementação que cumpre o processo agora descrito pode ser encontrada no Código 7.

Código 7: Tradução de alarmes IoT.nxt

```
@RequestMapping(value = "/alarms", method = RequestMethod.GET)
public ResponseEntity<Object> alarms() throws JSONException, URISyntaxException {
    //Clear previously translated alarms
    clearAlarms();
    //Getting the IoT.nxt Alarms
    String result = getAlarms();
    JSONArray arr = new JSONArray(result);
    JSONArray tmf_arr = new JSONArray();
    //Converting to TMForum and posting them on the TMForum AlarmManagement API
    for (int i = 0; i < arr.length(); i++) {
      tmf_arr.put(JsonUtils.mapAlarms(arr.getJSONObject(i),"iotnxt-tmforum"));
      postAlarms(JsonUtils.mapAlarms(arr.getJSONObject(i),"iotnxt-tmforum").
    toString());}
    //Redirecting to the AlarmManagement API alarm list
    URI tmfalarms = new URI("http://localhost:"+prop.getProperty("server.port")+"
    /alarmManagement/alarm");
    HttpHeaders httpHeaders = new HttpHeaders();
    httpHeaders.setLocation(tmfalarms);
    return new ResponseEntity<>(httpHeaders, HttpStatus.SEE_OTHER);}
```

Numa fase posterior foi adicionada à API a possibilidade de filtrar os alarmes da amostra com base no valor das seus atributos, fazendo com que seja feita o mapeamento de um grupo restrito de dados. A versão atualizada da função pode ser constada no Código 8.

Código 8: Excerto da função de tradução de alarmes com Filtros

```
@RequestMapping(value = "/alarms", method = RequestMethod.GET)
```

```
2  public ResponseEntity<Object> alarms(@RequestBody(required=false) String
     filters) throws JSONException, URISyntaxException {
     JSONObject aFilter = null;
4    if(!(filters == null)) {
     if(!filters.isBlank() || !filters.isEmpty()) {
6      aFilter = new JSONObject(filters);}}
     //Clear previously translated alarms
8    clearData("Alarm");
     //Getting the Iot.nxt Alarms
10   String result = getAlarms();
     JSONObject rJson = new JSONObject(result);
12   Object alarms = rJson.get("result");
     JSONArray arr = new JSONArray(alarms.toString());
14   if(!(aFilter == null)) {
     arr = filterData(arr, aFilter);}
16   JSONArray tmf_arr = new JSONArray();
         (...)
18   return new ResponseEntity<>(httpHeaders, HttpStatus.SEE_OTHER);}
```

Finalmente, sendo que pretende-se que se possa ser feita a tradução dos alarmes tanto por demanda do utilizador como recorrendo a *webhooks*. Ao utilizar este segundo tipo deste caso de utilização, a tradução acaba por ser feita em tempo real em vez de *on-demand* pois, ao invés de ser o utilizador/sistema a iniciar o processo de tradução cada vez que precisa, este irá ser invocado utilizando *Webhooks*. Os *Webhooks* são essencialmente *callbacks* HTTP definidos pelo utilizador, ou seja, na plataforma IoT é possível associar funções de API a eventos que possam ser acionados [33]. Neste contexto pretende-se associar os eventos relacionados com os alarmes à função de tradução em questão fazendo com que, sempre que esses eventos ocorram, essa função seja invocada, recebendo os alarmes que têm de ser encaminhados no corpo do pedido. A partir daí, o funcionamento é o mesmo do caso de utilização original, em que os alarmes recebidos serão traduzidos com base nas regras de mapeamento

definidas e o resultado será encaminhado para a API do TMForum. Comparativamente ao modelo *on-demand*, neste apenas serão feitas requisições de tradução quando há novas alterações ao invés de se realizarem vários pedidos de tradução enquanto não existem novidades nos dados, como acontece no outro modelo.

Assim, adaptou-se a função de tradução dos alarmes para ser utilizado no contexto de *webhooks*, tal como se pode ver no Código 9, em que a diferença é que essencialmente deixa de haver filtros nos parâmetros e os alarmes passam a ser obtidos diretamente do corpo do pedido.

Código 9: Excerto da função de tradução de alarmes para Webhooks

```
@RequestMapping(value = "/wh_alarms", method = RequestMethod.POST)
  public ResponseEntity<String> wh_alarms(@RequestBody String req_alarms ) throws
     JSONException, URISyntaxException {
    //Clear previously translated alarms
    clearData("Alarm");
    //Getting the alarms from the Request Body
    JSONArray arr = new JSONArray(req_alarms);
    JSONArray tmf_arr = new JSONArray();
    //Converting to TMForum and posting them on the TMForum Alarm API
    (...)
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
       LocalDateTime now = LocalDateTime.now();
    System.out.println( "The TMForum Alarm Management API was updated at " +dtf.
    format(now)+".");
    return new ResponseEntity<>(httpHeaders, HttpStatus.SEE_OTHER);}
```

4.4 Microsserviço - Resource Inventory Management API

Nesta fase do projeto foi feita uma análise à estrutura de dados da Resource Inventory Management API para projetar a maneira como seria possível mapear os dados das entidades provenientes da plataforma Resource Inventory Management API, foram definidas as regras de mapeamento e finalizou-se com a implementação da obtenção das entidades e da tradução das mesmas.

4.4.1 Análise da estrutura dos dados

Pretende-se utilizar esta API para alocar dados provenientes da plataforma IoT.nxt que são classificados como Entidades. As entidades são um tipo de dados que permite definir todos os recursos criados na plataforma, tais como contas de utilizadores, *sites* (representam as instalações de um conjunto de dispositivos sensores e atuadores virtuais ou físicos), dispositivos ou especificamente cada uma das suas propriedades, daí ser importante que a estrutura que os dados terão após a sua tradução seja o mais genérica possível de forma a poder representar todo o tipo de entidades.

Tabela 4.2: Atributos dos Recursos TMForum [1]

Atributos dos Recursos - Resource Inventory Management				
Obrigatórios	**Opcionais**			
name	activationFeature	administrativeState	attachment	category
hfref	description	endOperatingDate	note	operationalState
id	place	relatedParty	resourceCharacteristic	resourceRelationship
	resourceSpecification	resourceStatus		

Com base nos atributos possíveis de definir apresentados na Tabela

4.2, começou-se por analisar a estrutura das entidades provenientes da plataforma. Na Figura 4.7 encontra-se um excerto de JSON de uma entidade genérica da IoT.nxt em que figuram os atributos mais relevantes para serem mapeados para alguns dos atributos existentes nos dados da API.

Figura 4.7: Excerto uma entidade genérica da plataforma IoT.nxt

Tal como se pode constatar, os atributos presentes na figura são informativos em termos de caracterização da entidade a nível da sua identificação, nome, tipo, estado e relações de parentesco e filiação com outras entidades, resultantes da API *GetEntity*[34] existente na plataforma, ou seja, tem uma conjunto de propriedades que permitem detalhar qualquer tipo de dados de forma superficial.

Desta forma, à semelhança do que foi feito na Figura 4.5, foi realizada também uma planificação das regras de mapeamento para traduzir as entidades para recursos da TMForum que pode ser vista na Figura 4.8.

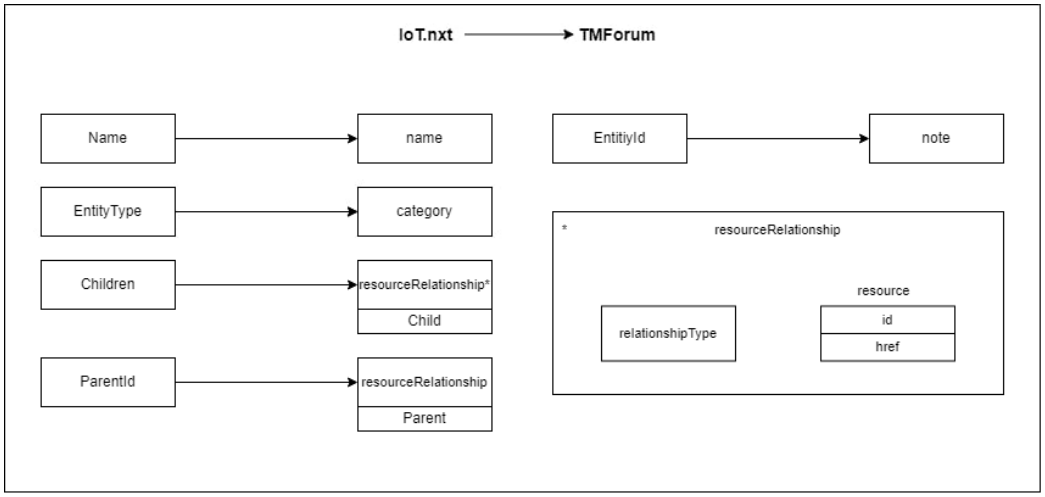

Figura 4.8: Planificação das regras de mapeamento de campos e de dados IoT.nxt para TMForum

No diagrama apresentado na Figura 4.8, a maior parte das ligações correspondem a mapeamentos diretos, ou seja, não há necessidade de alteração do formato nem do tipo do valor do atributo do dos dados da origem para que possa ser atribuído a um atributo dos dados de destino, tal como acontece com as ligações iniciadas em *Name*, *EntityType* e *EntityId*. No caso das ligações inciadas em *Children* e *ParentId* são apenas feitas alterações a nível do formato dos dados sendo que cada identificador do filho será representado por um JSON com o atributo *type*, com o valor que está associado ás caixas *resourceRelationship* (*Child* ou *Parent*), e o valor do identificador é atribuído ao atributo *id*.

4.4.2 Tradução de entidades IoT.nxt para TMForum

Nestas condições foi possível iniciar o processo de implementação dos meios necessários para realizar a tradução das entidades. O primeiro passo que se deu foi obter uma amostra de dados para aplicar os eventuais métodos de tradução, o que acabou por ser um desafio inicialmente pois

a API da IoT.nxt usada para obter os detalhes de cada entidade, *GetEntity*, apenas permite receber o *id* de uma só, retornando os seus detalhes. Sendo que seria necessário conhecer os identificadores das entidades todas que se quer traduzir antecipadamente, o que é impraticável visto que existem centenas de entidades na fonte dos dados, aproveitou-se o facto de cada entidade conter um atributo que lista todos os identificadores dos seus filhos para serem utilizados num novo pedido *GetEntity* que apenas necessita que se especifique o *id* da entidade em questão, tal como exemplificado na Figura 4.9.

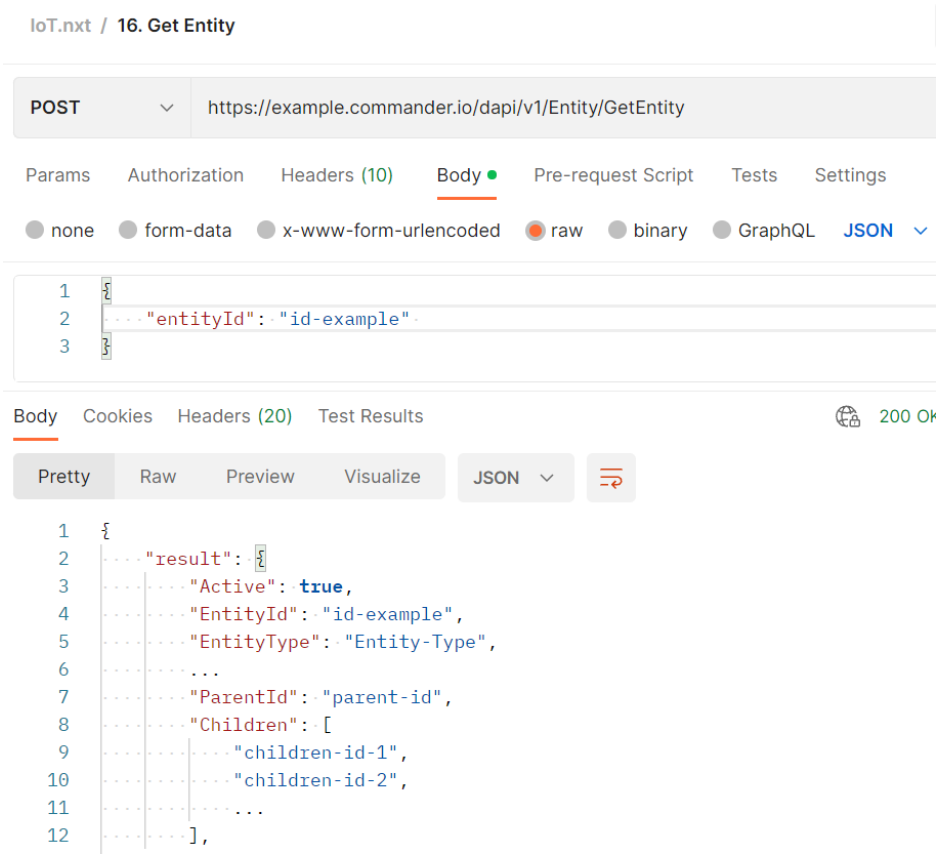

Figura 4.9: Exemplo do resultado de obtenção dos detalhes de uma entidade

Sendo que todas as entidades pertencem a uma entidade-raiz, que por sua vez também tem um atributo identificador e uma lista dos seus filhos

diretos, aplicou-se um método de procura em profundidade em que cada visita a uma nova entidade, recorrendo à da lista de filhos da sua entidade-pai, vai resultar na adição do JSON que contém os seus detalhes a um JSONArray, tal como mostra a Figura 4.10.

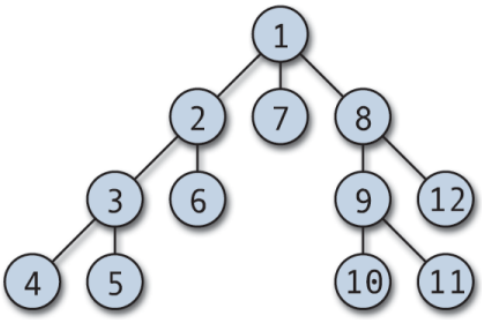

Figura 4.10: Visualização do Método de Procura em Profundidade

Associando a figura apresentada ao meio em que está a ser aplicada, o nó 1 representa a entidade raiz que após da obtenção dos seus detalhes através da utilização da API *GetEntity*, tal como foi exemplificado na Figura 4.9, pode-se verificar na sua lista de filhos os nós 2, 7 e 8. Tal como se pode concluir pela numeração dos nós, o 1º nó de cada nível é o utilizado para um novo pedido *GetEntity* e só se retorna a um nó de 2º nível após os detalhes de todos nós associados ao nó 2 terem sido guardados e assim o processo repte-se ás restantes entidades da plataforma.

A implementação deste processo iniciou-se replicando a utilização da API *GetEntity*, que foi exemplificada no Postman [31], utilizando uma função que recebe como parâmetro o *id* da entidade em questão e obtém os seus detalhes utilizando o *GetEntity* da IoT.nxt. A função em causa pode ser vista mais detalhadamente aravés do Códgio 10 apresentado.

Código 10: Função que replica a API *GetEntity*

```
1  public void getEntities(String entityId) throws URISyntaxException {
       HttpHeaders headers = new HttpHeaders();
       String GET_URL = ""+config.getProperty("entities.url");
3      if(config.getProperty("iotnxt.authorization")!=null)
         headers.set("Authorization", config.getProperty("iotnxt.authorization"));
5      headers.setContentType(MediaType.APPLICATION_JSON);
       RestTemplate restTemplate = new RestTemplate();
7      URI uri = new URI(GET_URL);
       String a = "{\"entityId\":\""+entityId+"\"}";
9      HttpEntity<String> requestEntity = new HttpEntity<String>(a,headers);
         String result = restTemplate.postForObject(uri, requestEntity,String.class)
       ;
11       getEntitiesFromRoot(result);
     }
```

Recorrendo ao método POST para fazer um pedido com o identificador da entidade como corpo da mensagem, como se pode ver na linha 10, é retornado a resposta do pedido que é guardada na variável *result* e que representa o JSON com os detalhes da entidade. De seguida, tal como se pode ver, é feita a invocação de uma outra função denominada por *getEntitiesFromResult* que recebe a variável *result* obtida anteriormente. A função *getEntitiesFromResult* é uma função indiretamente recursiva, por invocar a *getEntities* que por sua volta a invocar a primeira, fazendo um ciclo com o objetivo de aplicar o método de procura em profundidade apresentado na Figura 4.10.

Os detalhes desta função indiretamente recursiva podem ser vistos no Código 11.

Código 11: Função que aplica o Método de Procura em Profundidade

```
public void getEntitiesFromRoot(String rootEntity) throws URISyntaxException {
    JSONObject rJson = new JSONObject(rootEntity);
    Object rootEntityObject = rJson.get("result");
    JSONObject entityObject = new JSONObject(rootEntityObject.toString());
    entities.put(entityObject);
    String children = entityObject.get("Children").toString();
    String childrenSub = children.substring(1,children.length()-1);
    String[] childrenSplit = childrenSub.split(",");
    if(childrenSplit.length >0) {
    for(String child : childrenSplit) {
      if(child.length()>0) {
        getEntities(child.substring(1,child.length()-1));
      }}}}
```

Para fins de teste, adicionou-se a condição *if* para limitar o número de entidades retiradas da fonte de dados, limite que pode variar consoante a vontade do utilizador. Um excerto da versão de teste da função supracitada pode ser constada no Código 12.

Código 12: Função de teste que aplica o Método de Procura em Profundidade

```
public void getEntitiesFromRootTest(String rootEntity, int sampleSize) throws
    URISyntaxException {
    ...
    if(entities.length()<sampleSize) {
    if(childrenSplit.length >0) {
    for(String child : childrenSplit) {
      if(child.length()>0) {
        getEntities(child.substring(1,child.length()-1));
      }}}}}
```

À semelhança do que foi realizado para os alarmes, o diagrama apresentado na Figura 4.8 foi transcrito para o MongoDB para ser utilizado na função de mapeamento dos dados da plataforma. O resultado dessa transcrição pode ser notado na Figura 4.11.

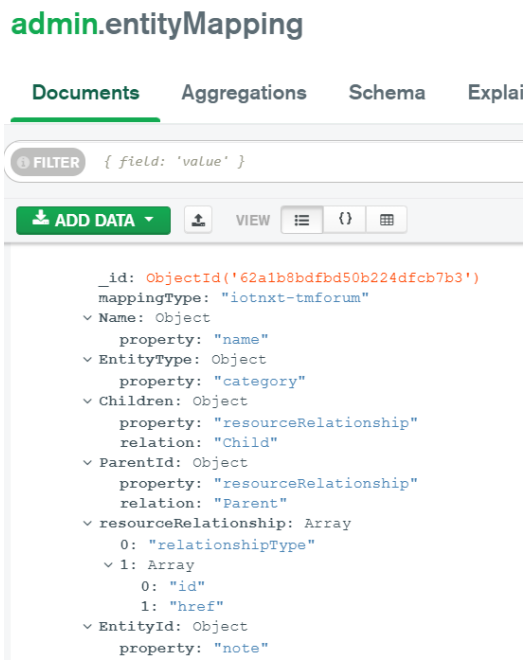

Figura 4.11: Definição das regras de mapeamento de entidades de campos e de dados em MongoDB

Com as regras de mapeamento das entidades disponíveis para ser acedidas através do MongoDB, é possível aplicar uma função de mapeamento à semelhança da função apresentada no Código 6. Tal como no caso dos alarmes, a maior parte dos tipos de mapeamento não necessitam de alteração do tipo de dados nem dos valores, o que é considerado como um mapeamento genérico que pode ser feito pela função *mapAlarms*, isto por ter sido implementada da forma mais genérica possível. Por essa razão, adaptou-se essa função para atender aos casos de mapeamento especial das entidades, como por exemplo o mapeamento dos atributos *ParentId* e do *Children*, acrescentou-se um parâmetro que permite indicar o tipo de dados que se pretende mapear e fez-se uma re-

nomeação da função para, mais uma vez, se tratar de uma função mais genérica.

Com o processo de mapeamento implementado, o processo de tradução inicia-se quando se acede ao *endpoint /entities* que foi implementado para iniciar o processo de procura a partir da raiz das entidades da plataforma, tal é mostrado no Código 13.

Código 13: Função que inicia o processo de tradução

```
@RequestMapping(value = "/resources", method = RequestMethod.GET)
public  ResponseEntity<Object>resources(@RequestBody(required=false) String
  filters) throws JSONException, URISyntaxException {
  JSONObject aFilter = null;
  if(!(filters == null)) {if(!filters.isBlank() || !filters.isEmpty())
    //Convert RequestBody to a JSONObject
    aFilter = new JSONObject(filters);}
  JSONArray arr = new JSONArray(getOrPostTest());
  if(!(aFilter == null)) {
    arr = filterData(arr, aFilter);}
  JSONArray tmf_arr = new JSONArray();
  //Converting them and posting them on the TMForum API
  for (int i = 0; i < arr.length(); i++) {
    JSONObject translatedResource = JsonUtils.mapData(arr.getJSONObject(i),"
  Entity","iotnxt-tmforum");
    tmf_arr.put(translatedResource);
    postResources(translatedResource.toString());}
  //Redirecting to the Resource Management API resources list
  URI tmfresources = new URI("http://localhost:9005/resourceInventoryManagement
  /resource");
  HttpHeaders httpHeaders = new HttpHeaders();
  httpHeaders.setLocation(tmfresources);
  return new ResponseEntity<>(httpHeaders, HttpStatus.SEE_OTHER);}
```

Tal como se pode ver pelo código supracitado, neste tipo de tradução foi também disponibilizada a hipótese de filtrar os dados da amostra ou da fonte de dados de forma a que apenas se traduza os dados que contenham atributos que contenham os valores dos filtros. Para isso, são usados os nomes dos atributos das entidades originais, atribuindo o valor de pesquisa, ou seja, caso se pretenda traduzir entidades que tivessem um nome relacionado com alarmes, definia-se o JSON de filtração assim: {Name: "Alarm"}. Ao ser enviado JSON como corpo do pedido *endpoint /resources* do *gateway*, é identificada a existência de filtros, é aplicada a função *filterData* que, tal como o nome sugere, acede à fonte dos dados, limitando para que apenas se traduza aquelas entidades contenham o termo "Alarm"no seu atributo *Name*. A função de filtração pode ser verificada com mais detalhe no Código 14.

Código 14: Função de filtração de entidades

```
private JSONArray filterData(JSONArray data , JSONObject dataFilters) {
    JSONArray filteredData = new JSONArray();
    for(int i=0; i<data.length();i++) {
      Iterator<String> keys = dataFilters.keys();
      while(keys.hasNext()) {
        String filter = keys.next();
        if(!data.getJSONObject(i).getString(filter).equalsIgnoreCase(dataFilters.
    getString(filter))) {
          break;}
        filteredData.put(data.getJSONObject(i));}}
      return filteredData;}
```

Tal como nos alarmes, foi implementada uma função para ser usada em contexto de *webhooks*.

4.5 Microsserviço - Geographic Site API

Apesar de se pretender ter como funcionalidade a capacidade de também traduzir dados referentes a informações geográficas provenientes da IoT.nxt e de se ter instalado o microsserviço que gere a criação e o armazenamento das dados geográficos traduzidos, não se prosseguiu com o processo de definição de regras de mapeamento e implementação da tradução dos dados pois atualmente não existem dados deste cariz que estivessem elegíveis para serem utilizados como amostra para projeto, não sendo possível o desenvolvimento desta fase.

Capítulo 5

Validação e Testes

5.1 Alarmes

Tal como projetado nos objetivos do projeto, pretendia-se fazer o teste da obtenção e tradução de alarmes tanto por demanda como por *webhooks*.

5.1.1 Tradução por demanda

Tal como referido em secções anteriores, a entende-se por ”tradução por demanda”o processo em que a tradução se inicia após o utilizador definir a fonte dos alarmes e aceder ao *endpoint* correspondente.

Para testar a API de tradução de alarmes implementada recorreu-se aos alarmes já existentes na plataforma de projetos e testes anteriores da empresa para garantir que se tratam de dados reais. A definição da fonte

dos dados realiza-se pela especificação *alarms.url*, presente no ficheiro de configurações do *gateway*, do *token* de autenticação *Bearer*, que se trata de um esquema de autenticação HTTP que envolve *tokens* de segurança para permitir o acesso aos dados, neste caso de teste de forma temporária, e do endereço e porto do MongoDB para se aceder ás regras de mapeamento definidas para os alarmes e guardar numa base de dados os alarmes obtidos, tal como se pode ver pelo Código 15.

Código 15: Excerto do ficheiro de configurações

```
mongodb.url=mongodb://localhost:27017
alarms.url=https://.../DAPI/v1/Alarm/GetHistoricalAlarmData
iotnxt.authorization=Bearer eyJhbG...
```

Apesar se ter utilizado uma amostra de 100 alarmes da plataforma IoT.nxt para testar o resultado do mapeamento, apenas se irá expor 1 único alarme de forma a garantir a confidencialidade dos dados provenientes de projetos da empresa. Após a utilização a obtenção das alarmes diretamente da amostra, um desses é o que está apresentado no Código 16.

Código 16: Documento JSON do alarme original da plataforma IoT.nxt

```
{"priority": "High",
"site": "69361 M_VELILLASANANTON_UMTS",
"logTime": "2022-05-30T10:24:59.191Z",
"description": "69361 M_VELILLASANANTON_UMTS - Indoor Temperature",
"alarmType": "Indoor Temperature Alarm",
"triggerName": "POC\69361 M_VELILLASANANTON_UMTS\Alarms\Indoor Temperature
    Trigger",
"timeActivated": "2021-08-23T11:47:00.096Z",
"lastActivated": "2021-08-23T11:47:00.096Z",
"state": "Inactive",
"mustAcknowledge": "Yes",
"acknowledged": "No"}
```

E, após a aplicação das regras de mapeamento e dos métodos de tradução apresentados no Capítulo 4, obtém-se o alarme apresentado no Código 17.

Código 17: Documento JSON do alarme traduzido


```
        "ackState": "unacknowledged",
2       "alarmRaisedTime": "2021-08-23T11:47:00.096Z",
        "alarmType": "environmentalAlarm",
4       "specificProblem": "Indoor Temperature Alarm",
        "probableCause": "Indoor Temperature Alarm",
6       "perceivedSeverity": "major",
        "sourceSystemId": "iot.nxt",
8       "place": [
            {
10              "role": "",
                "name": "69361 M_VELILLASANANTON_UMTS"
12          }
        ],
14      "alarmDetails": "69361 M_VELILLASANANTON_UMTS - Indoor Temperature",
        "state": "Inactive",
16      "alarmedObject": {
            "id": ""
18      },
        "externalAlarmId": "2
    c424c5915a6f07c054c6479cc5899e9de2b40d68c6cc491cc1e7abe2334509f",
20      "id": "f9bb9ee6-49d4-4c8a-83f5-114e44e1e763",
        "href": "http://localhost:8080/tmf-api/alarmManagement/v4/alarm/f9bb9ee6
    -49d4-4c8a-83f5-114e44e1e763"
```

Tendo em conta que o processo de mapeamento envolve a iteração por todos os atributos de cada alarme da amostra, sendo que isso envolve a utilização de ciclos *for*, uma das preocupações era a possibilidade da performance não ser ótima para completar a tarefa quando se tratassem de um número mais elevado de dados, por essa razão mediu-se o tempo decorrido desde a invocação do processo de tradução até à obtenção dos dados traduzidos.

Tabela 5.1: Tempos de Execução da tradução de alarmes

Tempo em segundos de Execução da Tradução de Alarmes (100 amostras)										
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Média**
1.509	1.447	1.293	2.286	1.609	1.234	1.006	1.892	1.6	1.117	1.493 s

A Tabela 5.1 permite fazer uma comparação dos diferentes tempos de execução para a tradução de 100 alarmes, concluindo que em média a duração do processo pode durar entre os 1-2 segundos para esta quantidade de alarmes. É importante frisar que os testes foram feitos utilizando um computador com as características apresentadas na tabela 5.2 para executar o projeto localmente. Com as especificações apresentadas, mais precisamente as do CPU e velocidades de processamento, dado ao cariz da implementação do projeto, quanto melhor o CPU, eventualmente maior será a velocidade de aplicação dos processos de tradução dos dados, sendo que a velocidade de acesso e envio de dados utilizando as API não sofrerá alterações por esse melhoria no computador. De forma geral, os resultados apresentados poderão ser melhorados ao executar o projeto num ambiente preparado para esse efeito.

Tabela 5.2: Especificações do PC utilizado nos testes

Especificações do PC	
Processador	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
RAM	16,0 GB (15,8 GB usable)
Armazenamento	Solid State Drive M.2 2280 512 GB

5.1.2 Tradução por *Webhooks*

A plataforma IoT.nxt disponibiliza a funcionalidade para configurar *webhooks* dentro da própria plataforma [35] mas, para que fosse possível fazer os testes necessários para validar a função implementada, seria preciso adquirir um ambiente na plataforma dedicado a este projeto, para não desregular os ambientes já existentes da empresa, processo que envolve tempo, custos e negociações com a empresa detentora da plataforma, logo acabou por ser um objetivo foi colocado em 2º plano até que se reúnam as condições necessárias para fazer o teste e certificar este caso como uma funcionalidade para utilização em futuros projetos da Celfocus .

5.2 Entidades

À semelhança dos alarmes, pretendia-se fazer o teste da obtenção e tradução de entidades tanto por demanda como por *webhooks*.

5.2.1 Tradução por demanda

No caso das entidades, para se poder fazer o teste, tal como referido no Capítulo 4, é necessário modular a amostra de dados utilizando o método de Procura em Profundidade para construir um JSONArray que armazena os documentos JSON que representam cada uma das entidades provenientes da plataforma IoT.nxt. Os testes realizados foram sempre feitos com amostras pequenas pois o processo de criação do JSONArray com as entidades todas disponíveis atrasa o processo todo, sendo que existem disponíveis mais de 11 mil entidades, o que levou aproximadamente 28 minutos utilizando o método mencionado.

Tal como nos alarmes, a fonte dos dados realiza-se pela especificação do campo *entities.url*, presente no ficheiro de configurações do *gateway*, do *token* de autenticação *Bearer* e do porto do MongoDB para se aceder ás regras de mapeamento definidas para as entidades e guardar numa base de dados os recursos traduzidos obtidos, tal como se pode ver pelo Código 18.

Código 18: Excerto do ficheiro de configurações

```
1 mongodb.url=mongodb://localhost:27017
  entities.url=https://.../dapi/v1/Entity/GetEntity
3 iotnxt.authorization=Bearer eyJhbG...
```

Mais uma vez, apesar se ter utilizado as entidades todas da plataforma IoT.nxt para testar o tempo de defnição do JSONArray e de se ter feito o teste de mapeamento utilizando uma amostra de 10 entidades, apenas se irá expor 1 única entidade de forma a garantir a confidencialidade dos dados provenientes de projetos da empresa. Após a utilização a obtenção das entidades utilizando o método de procura supracitado, um dos documentos JSON é o que está apresentado no Código 19

Código 19: Documento JSON da entidade original da plataforma IoT.nxt

```
1 {"result": {
```

```
        "RowVersion": "db1f3631-4b5c-43ac-beb7-3071d57e740d",
        "Active": true,
        "EntityId": "a3330a5c-feb3-4793-be43-9acd56c6feea",
        "EntityType": "GroupEntity",
        "Ancestors": [{"EntityId": "t000000040",
        "EntityType": "Client"},
        {"EntityId": "00000000-0000-0000-0000-000000000000",
        "EntityType":"Root"
        }],
        "Name": "QA Testing",
        "ParentId": "t000000040",
        "AccountId": "00000000-0000-0000-0000-000000000040",
        "Children": ["14dd8ea3-04bd-4f13-b78e-e9c2b9c69e5c",
            "a202500c-ce9a-4ee6-a3d5-bfd53474daab",
            "c47c3db5-16ab-4e6d-a8ed-b43af9ee5e8b",
            "5e96d858-8ab3-4d93-885c-b06102106d18",
            "e9c6d85b-1017-4a40-b643-a72262202079"],
        "CanHaveChildren": true,
        "Tags": [],
        "Level": 2},
    "messageType": "OK",
    "state": null,
    "sourceMessageID": "e847cfd434de43f2912f24173b71a789"}
```

E, pós a aplicação das regras de mapeamento e dos métodos de tradução de entidades apresentados no Capítulo 4, obtém-se o alarme apresentado no Código 20.

Código 20: Documento JSON da entidade traduzida

```
{"note": [{"text": "EntityId",
"id": "a3330a5c-feb3-4793-be43-9acd56c6feea"
}],"name": "QA Testing",
  "resourceRelationship": [ {"relationshipType": "Parent",
      "resource": {"id": "00000004",
      }},{"relationshipType": "Child",
      "resource": {
        "id": "14dd8ea3-04bd-4f13-b78e-e9c2b9c69e5c",
      }},{"relationshipType": "Child",
      "resource": {
        "id": "a202500c-ce9a-4ee6-a3d5-bfd53474daab",
      }},{
      "relationshipType": "Child",
      "resource": {
        "id": "c47c3db5-16ab-4e6d-a8ed-b43af9ee5e8b",
      }},{
      "relationshipType": "Child",
      "resource": {
        "id": "5e96d858-8ab3-4d93-885c-b06102106d18",
      }},{
      "relationshipType": "Child",
      "resource": {
        "id": "e9c6d85b-1017-4a40-b643-a72262202079",
      }}], "category": "GroupEntity",
  "id": "3759d046-ebcf-47f0-938d-dcc3491304c7"
}
```

As entidades obtidas na totalidade foram armazenadas no ficheiro à parte para que se pudesse obter as entidades de imediato e fazer a medição do tempo de execução dos processos de tradução. Fazendo esse teste, chegou-se a um valor de aproximadamente x segundos.

5.2.2 Tradução por *Webhooks*

Tal como nos alarmes, a situação descrita na Subsecção 5.1.2 mantém-se para as entidades, não tendo sido então possível fazer um teste legítimo usando as funcionalidades da própria plataforma.

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo centra-se na conclusão deste projeto, indicando as realizações mais relevantes no desenvolvimento do projeto. Após um breve resumo de toda a estrutura do documento, são descritos os objetivos que foram alcançados ao longo da evolução da solução apresentada assim como tarefas já definidas como trabalho futuro necessário.

O trabalho explicado neste relatório, contribui para um projeto criado pela CELFCOUS, que já foi e continuará a ser utilizado em projetos futuros da sua equipa de desenvolvimento de soluções IoT.

6.1 Recapitulação do Projeto

O desenvolvimento deste projeto foi tanto desafiante como enriquecedor. No início deste projeto tinha reduzido conhecimento sobre o mundo da Internet das Coisas e o seu potencial, o que acabou por se tornar numa

motivação para participar no estágio da Celfocus , tendo-me juntado à sua equipa de desenvolvimento de soluções IoT.

A fase inicial do trabalho requereu a obtenção de conhecimento em relação à forma como a Celfocus abordava o tema da IoT, conhecer e estudar o funcionamento das plataformas utilizadas pela empresa, comparando com outras, o que envolveu também a realização de formações/-testes que me permitiram ser autónomo aquando da sua utilização.

Após essa etapa, foi necessário conhecer melhor os tipos de dados da plataforma IoT.nxt propostos para serem traduzidos de forma a encontrar os melhores candidatos para utilizar como estrutura que seria utilizada para traduzir os alarmes, a qual acabou por ser as diferentes API do TMForum desde cedo, reunindo as melhores condições entre as diferentes outras API estudadas como possíveis alternativas para cada tipo de dados. As API do TMForum foram utilizadas como microsserviços auxiliares aos diferentes processos de tradução geridos pelo API *Gateway* desenvolvido em Spring Boot.

Uma das maiores dificuldades sentidas foi o estabelecimento dos meios ideais para definir e implementar as regras de mapeamento, de campo e de valor, que acabaram por ser feitas usando documentos JSON criados em MongoDB, sendo um ponto positivo visto que inicialmente previa-se que seria necessário usar plataformas diferentes para realizar os diferentes tipos de mapeamento.

Na etapa final do desenvolvimento, foram implementadas as funções que recorrem ás regras de mapeamento definidas, tanto para pedidos por demanda como por *webhooks*, foram desenvolvidos as diferentes caminhos de roteamento que unem as API do TMForum ao API *Gateway* desenvolvido e foram criados os *endpoints* para invocar os processos de tradução, reunindo-se assima as condições para testar a tradução das amostras de dados provenientes de projetos anteriores da empresa em IoT.nxt

6.2 Objetivos Atingidos

No início do projeto foram definidos alguns objetivos que se pretendiam atingir com o desenvolvimento deste projeto, sendo que nesta subsecção é feita a revisão sobre o que foi concluído e sobre o que ainda está por concluir.

Assim, concluiu-se os objetivos de:

- Desenvolvimento de um API Gateway em Spring Boot;
- Criação do microsserviço com a TMForum Alarm Management API;
- Criação de regras de mapeamento para traduzir alarmes IoT.nxt para TMForum;
- Tradução de alarmes IoT.nxt para TMForum;
- Gestão dos alarmes traduzidos recorrendo à interface gráfica do MongoDB;
- Criação do microsserviço com a TMForum Resource Management API;
- Criação de regras de mapeamento para traduzir entidades IoT.nxt para TMForum;
- Tradução de entidades IoT.nxt para TMForum;
- Gestão das entidades traduzidas recorrendo à interface gráfica do MongoDB;
- Criação do microsserviço com a TMForum Geographic Site API.

E ficou por concluir:

- Criação das regras de mapeamento e implementação de dados de informações geográficas por impossibilidade de acesso a dados desse cariz provenientes da IoT.nxt;

- Tradução invocada por *webhooks*.

6.3 Limitações e Trabalho Futuro

Uma das limitações sentidas durante o desenvolvimento do projeto foi o número de tipos de dados disponíveis para serem usados no contexto da tradução, visto que o TMForum disponibiliza muitas mais API para além das mencionadas ao longo do relatório, o que permitiria que mais tipos de dados fossem elegíveis para serem estandardizados pelos mesmos meios utilizados no caso dos alarmes e das entidades.

Como trabalho futuro prevê-se a adição de mais tipos de dados que possam ser traduzidos, pretende-se realizar novas traduções para as estruturas das API do TMForum a partir de dados de outras plataformas além da IoT.nxt e o desenvolvimento de uma interface gráfica que permita uma gestão mais simplificada dos dados traduzidos ao invés de se recorrer ao *Compass* do MongoDB, a interface gráfica que permite exploração e manipulação dos dados armazenados.

Bibliografia

[1] “Resource Inventory Management API User Guide,” 2020. [Online]. Available: https://www.tmforum.org/resources/standard/tmf639-resource-inventory-api-user-guide-v4-0/

[2] Celfocus, “CELFOCUS.” [Online]. Available: https://celfocus.com/home/

[3] TMForum, “TMForum,” 2021, accessed on 2021-11-24. [Online]. Available: https://www.tmforum.org

[4] “Alarm Management API User Guide,” 2020. [Online]. Available: https://www.tmforum.org/resources/specification/tmf642-alarm-management-api-user-guide-v4-0/

[5] “Geographic Site Management API User Guide,” 2020. [Online]. Available: https://www.tmforum.org/resources/standard/tmf674-geographic-site-management-api-user-guide-v4-0/

[6] B. T. India, “IoT Network,” 12 2021. [Online]. Available: https://businesstimesindia.com/index.php/2021/01/23/iot-impact-on-business/

[7] C. McClelland, “What is an IoT Platform?” 2017. [Online]. Available: https://www.iotforall.com/what-is-an-iot-platform

[8] R. J, “Green City PoC On Thingworx,” 2015.

[9] Statista, “IoT connected devices worldwide 2019-2030.” [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[10] IoT.nxt, “IoT.nxt Platfortm,” 2020, accessed on 2021-11-22.

[11] PTC, “Thingworx,” 2021. [Online]. Available: https://www.ptc.com/en/products/thingworx/

[12] R. Evolution, “Record Evolution,” 2021. [Online]. Available: https://www.record-evolution.de/en/home-en/

[13] R. E. GmbH, “IoT Platform Comparison.” [Online]. Available: https://www.record-evolution.de/en/blog/iot-platform-comparison-six-vendors-to-keep-watching-in-2021/

[14] Balena, “Balena,” 2021. [Online]. Available: https://www.balena.io/

[15] P. Industries, “Particle,” 2021. [Online]. Available: https://www.particle.io/

[16] TMForum, “TMForum Open APIs,” 2021, accessed on 2021-11-24. [Online]. Available: https://www.tmforum.org/open-apis/

[17] A. Hughes, “How TM Forum Open APIs are accelerating innovation delivery at Cerillion,” 5 2021. [Online]. Available: https://www.cerillion.com/Blog/May-2021/TM-Forum-Open-API-accelerating-innovation-delivery?aliaspath=%2fBlog%2fMay-2021%2fTM-Forum-Open-API-accelerating-innovation-delivery

[18] “Geographic Address Management API User Guide,” 2020. [Online]. Available: https://www.tmforum.org/resources/standard/tmf673-geographic-address-management-api-user-guide-v4-0-0/

[19] “Geographic Location API REST Specification,” 2018. [Online]. Available: https://www.tmforum.org/resources/specification/tmf675-geographic-location-api-rest-specification-r17-5-0/

[20] Oracle, “Oracle Field Service Cloud Integrating with Resource Management API 18A Integrating with Resource Management API,” 2018.

[21] IoT.nxt, “IoT.nxt Raptor Gateways.” [Online]. Available: https://community.iotnxt.com/docs/raptor-overview/

[22] IoT.nxt, “Edge Raptor.” [Online]. Available: https://community.iotnxt.com/docs/raptor/edge-raptor/

[23] C. Ltd, “Enterprise Open Source and Linux | Ubuntu.” [Online]. Available: https://ubuntu.com

[24] IoT.nxt, “raptorsnap.” [Online]. Available: https://snapcraft.io/raptorsnap

[25] VMWare, “Spring Boot.” [Online]. Available: https://spring.io/projects/spring-boot

[26] N. Trivedi, “Spring Cloud Gateway,” 6 2019. [Online]. Available: https://medium.com/@niral22/spring-cloud-gateway-tutorial-5311ddd59816

[27] VMWare, “Spring Tools 4.” [Online]. Available: https://spring.io/tools

[28] E. Foundation, “The Community for Open Innovation and Collaboration | The Eclipse Foundation.” [Online]. Available: https://www.eclipse.org

[29] MongoDB, “MongoDB Atlas: Cloud Document Database.” [Online]. Available: https://www.mongodb.com

[30] O. Foundation, “Node.js.” [Online]. Available: https://nodejs.org

[31] Postman, “Postman API Platform.” [Online]. Available: https://www.postman.com

[32] “INFORMATION TECHNOLOGY -OPEN SYSTEMS INTERCONNECTION -SYSTEMS MANAGEMENT: ALARM REPORTING FUNCTION ,” 1992.

[33] M. Guay, “what are webhooks?”

[34] IoT.nxt, “Iot.nxt Entity API,” 2020.

[35] IoT.nxt, “Webhook Integrations,” 2022. [Online]. Available: https://community.iotnxt.com/docs/commander/tools/webhooks/#webhooks-introduction

Apêndice A

Diagrama UML do Projeto

O diagrama de classe que reflete o projeto referente ao desenvolvimento da API Gateway encontra-se representado na Figura A.1.

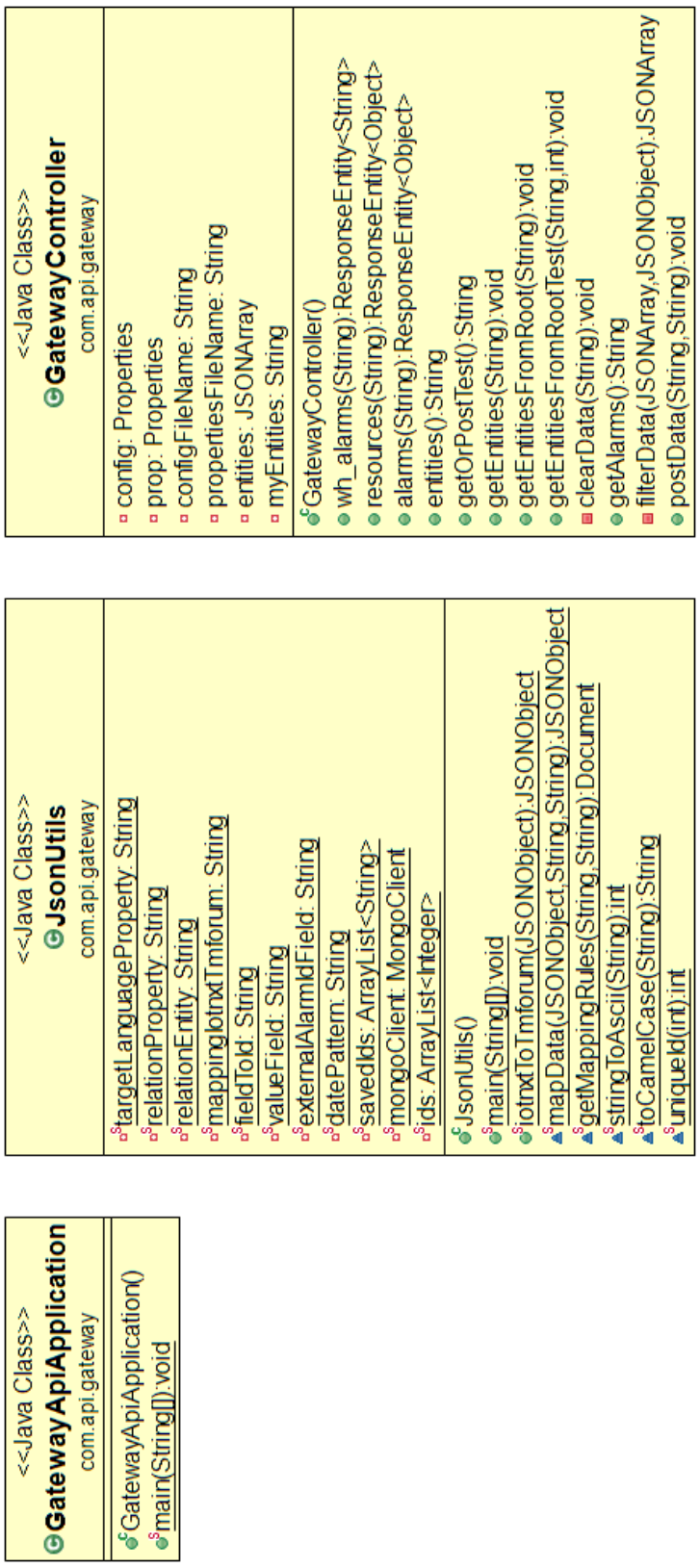

Figura A.1: Diagrama UML do projeto