João Miguel Peres Palma Neves

Licenciado em Engenharia Informática

Aquisição e Processamento de Dados de Energia

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Pedro Abílio Duarte de Medeiros, Professor Associado, Faculdade de Ciência e Tecnologia, Universidade Nova de Lisboa

Júri

Presidente: Dr. Carlos Augusto Isaac Piló Viegas Damásio
Arguentes: Dra. Susana Maria dos Santos Nascimento Martins de Almeida
Dr. Pedro Abílio Duarte de Medeiros

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2020

Aquisição e Processamento de Dados de Energia

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Agradecimentos

Esta tese não seria possível sem o apoio do meu orientador Professor Pedro Medeiros ou a equipa de desenvolvimento Redy, em especial o Head of Product Development da EDP Comercial, Tiago Antunes.

Resumo

Esta dissertação de mestrado enquadra-se no estágio profissional na empresa EDP Comercial para o qual fui seleccionado. Fui inserido na equipa de desenvolvimento do EDP Redy, um serviço que permite a gestão do consumo e produção eléctricos do cliente, bem como de actuação remota.

O meu trabalho focou-se em duas áreas do produto que precisavam de maior atenção. A primeira destas áreas foi o desenvolvimento de software especificamente para a versão do serviço Redy usado nos parques de uma operadora de telecomunicações. Trata-se de um projecto onde a operadora utiliza o serviço da EDP para monitorizar consumos energéticos. O meu trabalho passou também pela generalização deste software para poder ser utilizado noutros projectos com requerimentos similares.

Numa última fase vou desenvolver uma solução de testes automáticos usando a RobotFramework, uma framework de automação open source normalmente utilizada para testes automáticos. O meu trabalho nesta área teve o objectivo de diminuir o esforço da equipa de desenvolvimento no que toca Quality Assurance.

Palavras-chave: EDP Comercial Redy Casa Inteligente Automação Energia Consumo

Abstract

This master thesis stems from my professional internship with the company EDP Comercial, to which I was selected. I have integrated the development team for EDP Redy, an energy management solution that allows device control over the Internet.

My dissertation focuses on my contributions to the EDP Redy service in two areas where they were most useful. The first was on software development for Redy equipment specifically used on a project with a service provider. This is a project where the carrier uses the Redy service to monitor energy consumption. I will also work to generalize said software to enable it's reuse in other projects with similar requirements.

Lastly I developed the start of an automatic testing solution using RobotFramework, an open source framework used in test automation. The goal here was to relieve the development team in regards to *Quality Assurance*.

Keywords: EDP Comercial Redy Smart Home Automation Energy Consumption

Índice

Lista de Figuras

Listagens

Capítulo 1

Introdução

1.1 Motivação

A maior consciência de problemas ecológicos por parte do público em geral faz com que o mesmo tenha vindo a mostrar maior sede de informação. Ainda existe o hábito de desligar uma luz para poupar na conta da electricidade, quando na realidade a iluminação é dos factores que menos contribui para o consumo de uma casa de habitação. O público quer entender o que é a *potência contratada* e as *tarifas bi e tri horárias*.

As mesmas preocupações ambientais motivam uma secção mais pequena do público a fazer uso de soluções de produção de energia solar e adopção de carros híbridos ou eléctricos.

Como tal, existe um mercado crescente composto pelo público que não tem necessariamente conhecimento técnico, mas que quer monitorizar os seus consumos, controlar os gastos.

Tendo em conta o ambiente referido, a EDP Comercial oferece um serviço chamado EDP Redy [1]. Muitas vezes utilizado em conjunto com a solução de geração de energia solar da EDP, o EDP Redy endereça muitos dos pontos referidos como necessidades e desejos dos seus clientes. O serviço Redy disponibiliza:

- Monitorização de consumos gerais da casa com resolução de 15 minutos
- Monitorização de equipamentos ou conjuntos de equipamentos através de uma variedade de periféricos
- Actuação remota sobre equipamentos ou conjunto de equipamentos através de uma variedade de periféricos
- Controlo sobre a energia produzida com painéis solares

- Armazenamento de energia solar produzida em excesso através de uma bateria solar
- Alertas de consumo/produção fora do vulgar
- Eliminação de consumo standby dos equipamentos
- Programação do funcionamento dos equipamentos
- Dicas de ajuste de tarifa horária e potência contratada

O serviço Redy atinge estes objectivos através de um conjunto de periféricos instalados em casa do cliente. Estes periféricos comunicam com um dispositivo central (gateway) ainda em casa do cliente, que por sua vez interage com vários serviços cloud através da Internet.

A título de exemplo, um periférico a ler o consumo de um electrodoméstico envia dados por um protocolo de comunicação sem fios para a gateway. Esta envia uma mensagem para um serviço da Amazon Web Services [2], que guarda a informação relevante na cloud. Posteriormente o histórico de consumo daquele electrodoméstico pode ser consultado pelo utilizador através da app EDP Redy.

1.2 Contexto Mudança RITA -> OSGi

O meu estágio teve lugar durante uma mudança de tecnologia para o serviço. Anteriormente o software da gateway Redy era composto por uma framework PHP chamada RITA desenvolvida especificamente para o produto por uma empresa externa à EDP. Esta framework comunicava com uma plataforma que suportava todo o produto, desenvolvida por outra empresa externa. Este modo de funcionamento foi adoptado para facilitar o início de vida do produto mas tornou-se um factor impeditivo de desenvolvimento, pois a partir de um certo nível de complexidade do produto qualquer mudança implicava demasiado tempo e esforço de coordenação.

Assim foi decidido que devia ser feita uma mudança estrutural e o desenvolvimento do produto e plataforma devia ser feito internamente, com menor dependência possível de empresas externas. A nova framework Java OSGi[3] e a app foram desenvolvidas por empresas externas, mas toda a infraestrutura necessária para o funcionamento do produto seria desenvolvida pela equipa de desenvolvimento directamente, usando a plataforma Amazon Web Services. O custo de operação da infraestrutura baixou drasticamente, pois o pagamento destes serviços da Amazon são calculados em função do volume de utilização de cada serviço e sem custos antecipados de manutenção de um servidor, por exemplo.

O desenvolvimento da infraestrutura in-house tem o benefício de termos as duas empresas externas a depender do que é desenvolvido directamente pela equipa, ao invés do que acontecia antes onde uma mudança na framework RITA poderia depender de uma mudança na plataforma que a suportava e vice-versa, ambas desenvolvidas por empresas exteriores independentes.

O meu estágio tem lugar durante esta transição, altura em que já existem alguns clientes de teste a usar a nova plataforma e framework, mas a maioria ainda usa a plataforma e framework antigas. A equipa está repartida entre suportar o fim de vida destas e transportar toda a funcionalidade existente para um ambiente AWS.

1.3 Contribuições

O meu trabalho focou-se no melhoramento de duas áreas de maior carência do serviço Redy. Numa primeira fase, tendo em conta à mudança exposta na Seção 1.2, houve a necessidade de adaptar a framework OSGi no sentido de acomodar as necessidades de um cliente B2B. Por isto a minha primeira tarefa foi o desenvolvimento de um bundle Java OSGi para se poder fazer a transição de RITA para OSGi neste cliente também. O cliente em questão é uma operadora de telecomunicações que tem a necessidade de monitorizar o consumo eléctrico de equipamentos em parques de antenas. É necessário que o Redy monitorize os consumos dos equipamentos e envie os perfis de consumo periodicamente, em formato JSON, para um endpoint HTTP gerido pelo cliente. A grande dificuldade deste projecto são as condições atmosféricas adversas, a falta de pontos de internet nos parques e a dificuldade de comunicação através de Wifi.

Isto implica um estudo da framework OSGi em desenvolvimento pela empresa externa. Sendo eu o primeiro membro da equipa de desenvolvimento do Redy a programar directamente com a framework tive algumas dificuldades iniciais com a mesma. Existiam alguns tutoriais que eu implementei maioritariamente sobre o ambiente OSGi e gestão de dependências. Encontrei alguns problemas e o meu feedback melhorou alguns dos tutoriais. No entanto a documentação existente da framework em si era muitas vezes escassa e houve a necessidade de comunicar directamente com os programadores externos para resolver problemas e tirar algumas dúvidas.

A outra área abordada foi a de testes. Ao início do meu estágio cada membro da equipa era responsável pelo teste das suas próprias mudanças. Testes mais gerais eram feitos em equipa, com horas ou dias dedicados à tarefa. O meu trabalho foi o de criar um conjunto inicial de testes automáticos com o objectivo de poupar tempo à equipa quer nos períodos em que se realizam estes testes, quer durante o desenvolvimento. Com um conjunto grande e compreensivo de testes é possível à equipa identificar problemas e as suas causas mais cedo e produzir código mais seguro e de melhor qualidade sem um grande aumento de esforço. Para este efeito foi escolhida a RobotFramework [4] como framework base para desenvolvimento de testes pela sua flexibilidade, sintaxe e pelo facto de parte da equipa já ter tido contacto com a mesma.

1.4 Estrutura do documento

Este primeiro capítulo introdutório tem o objectivo de expor a motivação existente para criação do serviço EDP Redy e, consequentemente, para o trabalho desenvolvido durante

o meu estágio na EDP Comercial.

De seguida o capítulo dois tem o objectivo de oferecer algum contexto sobre sistemas de Energy Management e Home Automation no geral. O Redy é apenas um de centenas de produtos e soluções existentes, todas com o objectivos parecidos. É importante entender onde é que o serviço Redy se encaixa no mercado existente e como se compara com outras soluções.

O terceiro capítulo apresenta o serviço Redy em detalhe. As suas funcionalidades, os periféricos mais comuns, como estes comunicam com a gateway central e esta comunica com a infraestrutura cloud que suporta todo o serviço, bem como aspectos que podem ser melhorados.

O quarto capítulo descreve as decisões tomadas no que toca ao desenho das soluções. Expõe o funcionamento da framework Java OSGi a interacção da mesma com a infraestrutura que o suporta. Descreve também o funcionamento da RobotFramework e as razões pela qual foi escolhida.

O quinto capítulo expõe alguns detalhes da implementação dos meus contributos e contém explicação da organização dos ficheiros de ambas as soluções e excertos de código relevantes para as mesmas.

Finalmente apresento um balanço da minha prestação e do que aprendi enquanto colaborei com a equipa de desenvolvimento do EDP Redy.

Capítulo 2

Sistemas de Energy Management e Home Automation

2.1 Sistemas de aquisição e processamento de dados em consumos domésticos

Actualmente existem cada vez mais soluções de automação e controlo virados para o público em geral. Estas soluções têm por objectivo dar informação útil aos seus utilizadores e dar-lhes controlo dos seus aparelhos remotamente, através de regras personalizáveis, ou ambos. Desta forma os utilizadores são capazes de tornar mais eficiente o uso de energia dos seus equipamentos e cómodo o modo como estes operam.

Quer isto dizer que o utilizador de uma solução destas deve ser capaz de monitorizar o consumo dos seus equipamentos, bem como controlar sistemas de iluminação, estores, sensores de movimento ou de abertura de portas/janelas, electrodomésticos, smart TV's, ar condicionado, câmaras de vídeo vigilância entre outros, através de uma app ou aparelho único.

Neste capítulo pretende-se apresentar as funcionalidades esperadas e a arquitectura genérica deste tipo de soluções, bem como alguns exemplos de produtos e serviços que os implementam.

2.2 Funcionalidades

De uma solução de Home Automation é esperada uma certa gama de funcionalidades. Actualmente existe uma enorme gama de oferta deste tipo de produtos no mercado. Nesta secção quero expor algumas das funcionalidades geralmente esperadas destes serviços.

Medição de consumo Melhor visibilidade sobre o consumo energético da casa. Actualmente a informação que todos recebemos na factura da electricidade é o consumo geral da casa durante todo o mês. Este nível de granularidade não é suficiente para tirar conclusões muito úteis. Uma maneira de melhorar esta situação é tornar possível saber exactamente quanta energia foi gasta por cada equipamento. Isto é possível através de periféricos que fiquem entre cada equipamento e a sua fonte de energia, ou através da desagregação de cargas. Se o meu frigorífico é responsável por uma percentagem exagerada do consumo energético da casa pode querer dizer que está em fim de vida e deve ser substituído. Outra melhoria é a de aumentarmos a granularidade temporal e sermos capazes de ver o consumo energético geral ou de equipamentos específico em função do tempo. Poder ver o consumo energético por minuto, hora ou dia é mais interessante do que apenas saber quanto foi consumido durante um determinado mês. É, portanto, interessante que seja visível a quantidade de energia consumida por equipamentos específicos e sobre intervalos mais pequenos.

Actuação remota Ser capaz de actuar sobre os equipamentos sem ter de interagir fisicamente com os mesmos. É bastante interessante poder ligar, desligar e controlar equipamentos através de uma app no telemóvel.

Programação Possibilidade de definir regras e horários de funcionamento dos equipamentos. Dependendo da quantidade de sensores e dispositivos ligados numa rede pode ser possível automatizar imensos processos. Desde a automatização de rotinas matinais como abrir estores, ligar a máquina de café e as luzes se necessário, até trancar e destrancar a porta de casa com base na proximidade do utilizador, existe um numero muito grande de possibilidades.

Alertas Possibilidade de receber avisos quando existe comportamento fora do vulgar de forma a detectar um possível problema o mais cedo possível. Se o frigorífico deixar de consumir energia é um forte sinal que algo de errado se passa e quanto mais cedo o dono do frigorífico for notificado, melhor. Se tivermos sensores de movimento a serem desencadeados quando não devia estar ninguém em casa também pode ser causa para alarme.

Compatibilidade Com a situação actual do mercado existem dezenas ou até centenas de empresas que oferecem soluções de Home Automation. Existem protocolos de comunicação estandardizados cujo objectivo é deixar que periféricos de empresas diferentes comuniquem e trabalhem juntos. Claro que embora esta ideia seja óptima para o consumidor, muitas vezes as empresas que produzem estas soluções têm mais a ganhar em manter um ecossistema exclusivo e muitas vezes o principio da compatibilidade não é seguido.

Medição de produção energética Se uma casa tiver em funcionamento uma forma de produção de energia, como painéis solares, pode ser interessante poder obter informação sobre o seu funcionamento.

2.3 Mercado Actual

Actualmente existem inúmeros produtos de imensas empresas que podem ser considerados da categoria de Energy Management e Home Automation. Desde pequenos sensores de temperatura, a home assistants como a Amazon Echo. O modo mais simples de compreender toda a oferta no mercado actual é olhar para certas características diferenciadoras dos produtos de forma a mais facilmente perceber as suas semelhanças e diferenças.

Os produtos que se destacam mais facilmente são os Home Assistants Amazon Echo [5], Google Home [6] e HomePod [7], desenvolvidos respectivamente pela Amazon, Google e Apple. Estes home assistants são muitas vezes usados como ponto central de controlo para todos os periféricos numa casa. Amazon Echo e Google Home são os que oferecem maior compatibilidade com devices de outras marcas. O HomePod continua a tradição da Apple de manter o seu ecossistema fechado, trabalhando apenas com algumas marcas e dispositivos. Estes home assistants são aparelhos equipados com microfones, colunas e capacidade de comunicação por Wifi e em alguns casos Zigbee [8] e outros protocolos de comunicação sem fios. Podem ser usados apenas como colunas e home theaters, são capazes de ouvir e responder a comandos de voz mas a sua verdadeira utilidade é funcionarem como ponto único de controlo de todos os dispositivos de uma habitação (admitindo que os dispositivos em questão são compatíveis). São capazes de comunicar com os sensores que possam existir e possibilitam a programação de regras ou rotinas.

Existe outro tipo de dispositivo que serve funções parecidas mas não é considerado um Home Assistant. Estes são chamados Hubs. Por norma não têm colunas ou microfones embutidos e o seu único propósito é o de agregar informação e controlar outros dispositivos. Nesta categoria a oferta é vastamente maior e é onde o mercado começa a ficar caótico. Com muitas marcas, com muitos produtos e muitas versões do mesmo produto é sempre difícil garantir compatibilidades antes da compra. Actualmente alguns dos Hubs mais populares são o Samsung SmartThings Hub [9] e o Hubitat Elevation[10].

A próxima distinção importante é entre os dispositivos que necessitam de um Hub para funcionar e os que se ligam directamente à rede Wifi e podem ser controlados através de uma app. O melhor exemplo disto são os os produtos da Xiaomi Smart Home [11]. Parte dos dispositivos podem operar sozinhos ligando-se directamente à rede wifi e sendo controlados por uma app. Outros, normalmente por serem apenas capazes de comunicar por Zigbee, precisam de um Hub para serem úteis. Tipicamente dispositivos de uma marca só funcionam com o Hub dessa marca a não ser que seja especificado que é compatível com outros Hubs específicos. Vivemos um período em que o mercado de Home Automation está a sofrer constante inovação e volatilidade, com imensa oferta de

soluções e ecossistemas. No próximo capítulo vou apresentar o EDP Redy, a tentativa da EDP em penetrar este mercado.

Capítulo 3

EDP Redy

3.1 Apresentação do Redy

O Redy[1] é um produto que já existe há alguns anos e já tem uma infraestrutura considerável a suportá-lo. Como já foi explicado, o serviço EDP Redy permite a monitorização do consumo e produção de energia, actuação e programação de dispositivos e envio de alertas de falha de energia ou consumo irregular. Fá-lo através de periféricos instalados no quadro eléctrico ou onde necessário em casa do cliente. Estes periféricos estão em comunicação com a Redy Box ou Gateway, um dispositivo central que agrega a informação dos periféricos. Este dispositivo agrega a informação dos periféricos e comunica através da internet com o Redy Technical Support (RTS) e um conjunto de serviços da Amazon Web Services (AWS). O RTS é utilizado pela equipa de suporte da EDP para diagnosticar e possivelmente resolver problemas ou dificuldades que os clientes possam encontrar no uso do serviço. Mais detalhe em 3.1.3 RTS. A AWS é usada maioritariamente pela equipa de desenvolvimento. Os dados dos periféricos, casas e utilizadores são guardados na DynamoDB, controlo de acessos é feito pelo Cognito, comunicações com gateways são tratadas pela IOT e pela API Gateway, enquanto Lambdas fazem a ponte entre todos estes serviços. Mais detalhe no ponto 4.1.3 Serviços AWS.

3.1.1 Periféricos

Redy Box ou **Gateway** é responsável por comunicar com todos os dispositivos instalados na casa dos clientes. Esta comunicação serve para reportar consumo, produção e estado dos equipamentos para a AWS, bem como por em prática os comandos e programações definidos pelos utilizadores.

Figura 3.1: EDP Redy Gateway

Redy Modem Equipamento pequeno ligado a contadores inteligentes ou smart meters permite que a Gateway recolha informação sobre o consumo geral da habitação. É compatível com alguns modelos de contadores inteligentes em Portugal, como é o caso da Redy Box.

Figura 3.2: EDP Redy Modem

Redy Meter Módulo com capacidade de medição de energia de três circuitos monofásicos ou um trifásico, e actuação local e remota em outros dois circuitos. Este dispositivo é instalado no quadro geral da habitação e possibilita a monitorização isolada de, por exemplo, a energia consumida no sistema de iluminação da toda a habitação. Em paralelo, tem a capacidade de actuar em dois circuitos o que permite controlar remotamente equipamentos que se encontrem a maiores distâncias da habitação, como bombas de piscinas, iluminação de jardim ou equipamentos de rega.

Figura 3.3: EDP Redy Meter

Redy Switch Interruptor inteligente com capacidade de controlo remoto, ideal para controlo de equipamentos como caldeiras ou para colocação em tectos falsos e caixas de derivação para controlo de iluminação. O Redy Switch é um dispositivo de pequenas dimensões cuja finalidade é o controlo de um circuito ou equipamento, não reportando assim consumos para a Gateway. Em caso de necessidade pode ser actuado localmente através do botão físico mas, como o Redy Meter, permite actuação remota e programação.

Figura 3.4: EDP Redy Switch

Redy Plug É uma tomada inteligente para ser colocada individualmente em cada equipamento com o objectivo de recolher medidas de energia consumida ou produzida e ainda de controlar remotamente esse mesmo equipamento. Pode também ser ligado ou desligado localmente através de um botão físico.

Figura 3.5: EDP Redy Plug

Redy Plug Solar Uma tomada inteligente similar à Redy Plug mas com capacidade de medir não só consumo energético como produção. As soluções de geração de energia solar vendidas ao público por parte da EDP estão sempre acopladas ao Redy, sendo esta plug usada para monitorizar o consumo dos painéis solares.

Redy Plug AC Uma tomada inteligente com as mesmas funcionalidades da Redy Plug mas com a capacidade adicional de controlar uma unidade de ar condicionado através de infravermelhos. Tem de ser colocada em linha de vista com a unidade AC que se pretende controlar.

Figura 3.6: EDP Redy Plug AC

Redy Thermostat Termostato que permite controlar o sistema de aquecimento central da habitação de modo a permitir o controlo da temperatura no interior da habitação remotamente.É também possível fazer agendamentos para aquecer ou arrefecer de acordo com o desejado. É alimentado por 2 pilhas AA de 1,5V com uma vida útil até 18 meses.

Figura 3.7: EDP Redy Thermostat

Redy Door/Window Sensor Sensor magnético para ser instalado em portas e janelas com o objectivo de detectar se estas foram abertas ou fechadas. Tem também a

capacidade de medir a temperatura da divisão até a um máximo de 50ºC. A fonte de energia são 2 pilhas AAA que podem durar até 2 anos e reporta informação de estado e de medidas de temperatura a cada 2 minutos.

Figura 3.8: EDP Door/Window Sensor

Redy Motion Sensor O sensor de movimento é um sensor de infravermelhos que permite detectar movimentos até a um raio de 6 metros em linha de vista. Pode ser colocado em qualquer superfície mas também pode ser fixado na parede ou tecto. Tem ainda a capacidade de medição de temperatura da divisão desde 0 a 50ºC. Reporta informação de estado e de medidas de temperatura a cada 2 minutos e é alimentado por uma pilha CR123 que terá uma vida útil até 3 anos.

Figura 3.9: EDP Motion Sensor

Redy Smoke Sensor Sensor óptico de detecção de fumo que lança um alerta sonoro. Tem ainda a capacidade de medição de temperatura desde 0 a 50ºC e é facilmente fixado no tecto de cada divisão. Reporta informação de estado e de medidas de temperatura a cada 5 minutos e é alimentado por uma pilha CR123 com uma vida útil até 5 anos.

Figura 3.10: EDP Smoke Sensor

Redy Flood Sensor Sensor para detecção de fugas de água, infiltrações ou mesmo inundações devido a condições atmosféricas. O sensor é colocado ao nível do chão e quando detecta a presença de água, um alerta sonoro é lançado. Tem ainda a capacidade de medição de temperatura desde 0 a 50ºC. Reporta informação de estado e de medidas de temperatura a cada 5 minutos e é alimentado por uma pilha CR123 com uma vida útil até 5 anos.

Figura 3.11: EDP Flood Sensor

3.1.2 Gateway

A gateway esteve até recentemente a correr uma aplicação em PHP que reportava a um portal desenvolvido por uma empresa externa. Actualmente surgiu a necessidade de deixar de usar este portal e passar a explorar serviços da AWS. A aplicação PHP foi substituída por uma framework em Java chamada OSGi para simplificar a integração com os ditos serviços. Nesta secção explicarei brevemente o funcionamento interno da gateway

OSGi

O OSGi (Open Service Gateway Initiative) [3] é uma framework Java para desenvolvimento de aplicações modulares. A especificação desta framework prevê entidades independentes chamadas bundles ou plug-ins e como estes podem ser criados e apagados dinamicamente, bem como a possibilidade de estes bundles publicarem e subscreverem serviços uns dos outros.

Na gateway correm vários destes bundles que disponibilizam vários serviços. A maioria funciona como abstracção para comunicação com periféricos, implementação de regras, actuação, gestão de variáveis de histórico, entre outros.

Comunicação

A comunicação com os periféricos é controlada por bundles OSGi já desenvolvidos. Estes abstraem a comunicação com os vários tipos de periféricos, uniformizando a forma como são acedidos. Na prática, a comunicação entre a gateway e os periféricos são efectuados por 2 protocolos:

Zigbee [8] É o protocolo sem fios mais utilizado no mundo da Internet of things. Possibilita a criação de uma rede local de que liga todos os aparelhos, mas em que cada um gasta muito pouca energia e só consegue comunicar directamente com outros aparelhos a curtas distâncias. Por utilizar muito pouca energia é possível ter equipamentos como sensores de movimento (que não estão ligados à corrente) a fazer parte desta rede durante mais de um ano sem ser necessário trocar a sua fonte de alimentação.

PLC *Power Line Comunication* é um protocolo usado para envio de dados sobre um condutor que também transporta energia eléctrica. Porque a frequência do sinal eléctrico está entre 50-60Hz é possível sobrepor a este sinal um outro cuja frequência é muito mais elevada, rondando os 1-30MHz. Assim é possível usar estas altas frequências para envio de dados sem levantar problemas aos equipamentos que estão ligados à mesma rede eléctrica.

Com estes dois protocolos todos os periféricos do serviço EDP Redy são capazes de comunicar entre eles e com a gateway, podendo enviar relatórios de consumo, produção, ou receber comandos.

BusyBox

Esta framework OSGi corre em cima do BusyBox[12], uma aplicação com um executável muito pequeno desenhada para *embedded systems* que oferece uma interacção fácil com o sistema operativo por meio de comandos Linux.

Deploy

O serviço EDP Redy já está distribuído pelos clientes. Quer isto dizer que todos os updates têm de ser feitos remotamente. A comunicação por SSH é gerida pela própria framework OSGi. Quer isto dizer que não é possivel através de uma ligação SSH substituir os ficheiros da framework, pelo que se a mesma pára de correr, a ligação SSH também se perde. Assim, o modo de desenvolvimento e deploy de software terá de ser feito através do Redy Technical Support (RTS).

Depois de uma nova versão do bundle ser escrita e testada numa gateway local é gerado um ficheiro .jar. Este é enviado para uma ou mais gateways no terreno. Para contornar o problema da ligação falhar quando a framework é terminada são usados um conjunto de scritps e o crontab, um pequeno programa contido no BusyBox que trata de correr certos scripts a certas datas ou em intervalos regulares. Através deste programa os ficheiros relativos à framework são substituídos e a gateway sofre um reboot. Ao acordar a framework é corrida automaticamente e a gateway está a correr a sua nova versão. Se por acaso existir um problema que impeça esta nova versão de funcionar, então a única solução é ir ao terreno reverter as mudanças, porque sem a framework a correr é impossível a comunicação com o RTS.

3.1.3 Infraestrutura

Como mencionado no capítulo 1.2 o serviço EDP Redy encontrava-se no processo de mudança de tecnologias. Antes da adopção dos serviços da Amazon a gateway Redy corria uma aplicação em PHP e era suportada pelo Redy Technical Support e uma plataforma desenvolvida por uma empresa externa.

O objectivo é passar todos os clientes para a versão Java OSGi, portanto o RTS passará a ser uma plataforma capaz de fazer operações de manutenção e inspecção e fazer de interface com os serviços centrais da EDP, alguns dos quais descritos na secção seguinte.

Esta transição apresenta oportunidades de melhoria na organização da equipa de desenvolvimento, visto que baixa drasticamente o custo de adoptar uma metodologia de Continuous Itegration Continuous Development. Com vista a este objectivo foi desenvolvida uma bateria de testes Robot Framework. A escolha desta framework é especificada na secção 4.2.3.

RTS

O *Redy Technical Support* é uma plataforma de suporte técnico e é utilizada para gerir e interagir com os vários equipamentos EDP Redy usado pelos colaboradores da EDP como o Contact Center, instaladores do serviço e a equipa de desenvolvimento. Tem toda a informação sobre a forma como estão configurados os vários dispositivos e por isso é também utilizada na resolução de problemas com os mesmos. Pode ser usado para fazer operações em massa nas gateways, inspeccionar a rede Zigbee de uma determinada habitação, actuar sobre periféricos, fazer update ao software usado nas gateways e abrir comunicação directa com a gateway através de *SSH*.

AWS

Todavia o futuro da infraestrutura do Redy está na Amazon Web Services [2]. As gateways a correr OSGi que já estão em casa de clientes são suportadas pela Amazon, embora a comunicação com o RTS seja mantida de forma a que o software da gateway seja transparente para outra equipas da EDP como a de suporte ou de instalação. Mais detalhe sobre os serviços utilizados pelo Redy na Secção 4.1.3.

3.1.4 App

O serviço EDP Redy inclui uma aplicação. Este é o meio principal de interacção com o utilizador. A app disponibiliza as seguintes funcionalidades:

- Consultar consumo instantâneo da casa e de cada equipamento monitorizado individualmente
- Consultar o histórico de consumo da casa e de cada equipamento monitorizado individualmente
- Consultar o estado dos equipamentos
- Controlar equipamentos
- Programar equipamentos com base em regras
- Receber alertas

Alertas

Um utilizador do serviço EDP Redy pode escolher receber notificações quando algo fora do comum acontece. Alertas mais comuns são os de não-consumo de um equipamento como o frigorífico, falha de energia geral e consumo elevado de um equipamento.

3.2 Possíveis melhorias

Durante o meu estágio cada elemento da equipa de desenvolvimento estava a trabalhar para melhorar o produto na sua área. Sendo o Redy um produto complexo, com muitos clientes activos e em mudança de tecnologia e infraestrutura, havia a necessidade de fazer frente a vários problemas.

A app é um dos pontos mais importantes, pois é o elemento com o qual os clientes interagem directamente. Ter uma app a funcionar sem problemas em iOS e Android, em incontáveis dispositivos e com boa performance é sempre um desafio. A integração do serviço com assistentes virtuais como Amazon Alexa e Google Home poderiam ser interessantes, visto que o público alvo do Redy são clientes com algum conhecimento técnico.

O Redy não é um produto acabado, e como tal parte do esforço da equipa de desenvolvimento está focado em aumentar as capacidades do mesmo. Está no plano da EDP Comercial expandir a funcionalidade do Redy com novos periféricos e com novas soluções que facilitem o uso do mesmo. Uma possibilidade para tal é a desagregação de consumos. Este processo consiste em analisar muito pequenas variações de corrente eléctrica quando um electrodoméstico é ligado e durante o seu funcionamento. A partir desta informação é possível descobrir qual o equipamento que se está a utilizar. Esta análise da fingerprint de cada equipamento permite a produtos como a Voltaware monitorizar os consumos de todo e cada equipamento e luz de uma habitação necessitando apenas de instalar um pequeno dispositivo no quadro eléctrico. De igual forma o desenvolvimento de novos periféricos pode acrescentar valor ao produto. Controlo de estores ou um sistema de iluminação inteligente podem cobrir muitos casos de uso e trazer novos clientes.

Em certos cenários faz sentido considerar a possibilidade de nos afastarmos do modelo Business to Consumer (B2C) e olharmos para a vertente Business to Business (B2B). O Redy pode ser vendido a empresas que precisem de uma solução de monitorização de consumos e/ou actuação remota, e pode ser implementado em massa onde necessário.

Estando o Redy a mudar para uma framework em Java abre a porta a que, com as condições certas, se possam organizar eventos de comunidade como hackathons de maneira a potenciar inovação no campo de Energy Management e Home Automation. Para este efeito a framework Java OSGi que está a ser desenvolvida tem de ser devidamente documentada e testada de forma a ser fácil a sua manipulação.

Actualmente cada membro da equipa é responsável por testar o próprio código quando este é modificado. Para fazer testes end-to-end são reservadas horas ou dias de trabalho. Este processo pode ser melhorado com a introdução de métodos de Continuous Integration e Continuous Delivery [13]. Refiro-me a soluções que automatizam o processo de compilação, teste e deploy de projectos. Para atingir este fim é necessário uma ferramenta como Jenkins[14] ou AWS CodePipeline [15], uma bateria de testes automáticos para serem corridos sempre que o o projecto sofre alterações e uma mudança de método de trabalho por parte de toda a equipa de forma a tomar partido dos benefícios desta

metodologia. São esses benefícios o aumento da qualidade do código (porque este está constantemente a ser testado), a eliminação do tempo anteriormente gasto a juntar a equipa para fazer testes end-to-end e maior facilidade e rapidez em encontrar e resolver problemas, entre outros.

Estes são alguns exemplos de áreas onde o investimento de recursos pode ser proveitoso para o produto.

Capítulo 4

Organização da solução

Neste capítulo pretendo apresentar com detalhe as decisões que tomei durante o meu estágio, no que toca ao desenho da solução. Para isto vou partir o meu trabalho em duas secções. A primeira parte incide sobre o bundle Java OSGi e a segunda incide sobre a bateria de testes RobotFramework desenvolvida.

4.1 Java OSGi

4.1.1 Condições no terreno

Uma operadora de telecomunicações está a usar o serviço EDP Redy para monitorizar consumos energéticos em vários parques espalhados pelo país. Estes parques apresentam condições que dificultam o funcionamento correcto do serviço. Cada parque tem dezenas de gateways e periféricos expostos a altos níveis de humidade e com dificuldades de ligação à internet por estarem directamente debaixo da torre de comunicação. As gateways de todo o serviço EDP Redy corriam uma aplicação em PHP. Recentemente surgiu a necessidade de ter as gateways a correr uma framework Java OSGi, de maneira a conseguir integração com vários serviços da AWS. Com a descontinuação da aplicação em PHP foi o meu trabalho desenvolver um bundle OSGI especificamente para este cliente B2B. O seu propósito é receber os dados de consumo provenientes dos periféricos ligados a cada gateway, tratá-los e envia-los por HTTP para um endpoint previamente definido. Tive de ter em conta também as condições em que o serviço ia operar de modo a tentar atenuar os problemas vindos das condições em que o equipamento operava.

O objectivo do contracto entre a empresa de telecomunicações e a EDP é de monitorizar consumos em parques de comunicação. A principal dificuldade destes parques para o serviço Redy é a dificuldade de comunicação sobre Wifi. Todos os parques têm pequenos conjuntos de equipamentos, e cada conjunto é composto pelos equipamentos

cujos consumos devem ser medidos, os periféricos necessários para essa medição ligados a uma gateway Redy, e um router Wifi que possibilita a comunicação da gateway. A entrega dos perfis de consumo é feita através de um POST HTTP. A máquina responsável por receber esta mensagem pertence à empresa contratante.

Durante o meu estágio foram discutidas formas de lidar com a situação. Experimentalmente chegou-se à conclusão que um reset periódico aos routers diminuía o numero de falhas, mas mais nenhuma solução implementável eliminou o problema. Quer isto dizer que de forma a diminuir a carga na rede, seria importante limitar ao máximo o número de mensagens enviadas por cada gateway, e uniformizar o envio das mesmas no tempo.

4.1.2 Organização da Framework

A Framework OSGi tem como ideia base a existência de bundles, entidades modulares que implementam um certo serviço. No contexto do EDP Redy estes serviços podem ser responsáveis por tarefas como acesso ao historial de dados, tratamento de eventos originados pelos periféricos, aplicação de regras definidas pelo utilizador, comunicação com o contador, comunicação com a AWS, entre outros. Cada um destes bundles, ao ser iniciado, regista-se na framework e assim anuncia a sua presença a todos os outros bundles. Esta abordagem deixa que cada bundle inicie, pare ou modifique o seu comportamento dinamicamente consoante o contexto de execução.

O bundle que desenvolvi para o cliente B2B espera que três bundles específicos iniciem o seu funcionamento, pois os serviços prestados pelos mesmos são essenciais ao desempenho das suas funções. São os bundles em causa os responsáveis pelo acesso ao histórico de dados de consumo, pela notificação de entradas e saídas de novos periféricos e pelo acesso às informações dos dispositivos.

4.1.3 Serviços AWS

O serviço EDP Redy faz uso de vários serviços da Amazon Web Services. Pela sua facilidade de uso, confiabilidade, escalabilidade e custo reduzido. Um dos bundles da framework é responsável pela comunicação entre a gateway e a AWS. Estes são os serviços utilizados pelo serviço EDP Redy:

IOT [16] A sigla representa a *Internet of Things*. É o serviço da AWS com qual a gateway comunica. Cada gateway é representada como uma "thing"neste serviço e tem o seu próprio certificado de segurança e chave privada. É possível monitorizar o fluxo de mensagens de cada gateway e a integração com outros serviços AWS é muito simples.

API Gateway [17] Serviço da AWS que permite criação e disponibilização de API's de forma simples, rápida e em escala mundial se necessário.

DynamoDB [18] Base de dados key-value e orientada ao documento, que oferece performance na ordem do milissegundo a qualquer escala. Toda a informação relevante para o funcionamento correcto do serviço EDP Redy é guardada aqui.

Lambdas [19] Pequenas funções que juntam todos os serviços acima descritos. Cada endpoint de uma API gerada pela API Gateway chama uma Lambda. Esta função, por sua vez, pode consultar informações na DynamoDB, usar o IOT para enviar uma mensagem a alguma gateway ou até chamar outra Lambda

Clowdwatch [20] Ferramenta que ajuda a monitorização das outras ferramentas. Aqui podem ser consultados alertas, logs, métricas de performance, entre outros.

Cognito [21] Serviço que gere autenticação e controlo de acessos de utilizadores de forma segura.

4.1.4 Lógica da solução

A função do bundle desenvolvido é relativamente simples. Deve enviar a informação de consumo energético de todos os dispositivos emparelhados, em formato JSON, para um endpoint HTTP. A data a partir da qual deve enviar informação histórica, o endereço do endpoint e algumas outras informações estão definidas num ficheiro de configuração que é lido quando o bundle é activado.

Para os efeitos desta explicação, defina-se um perfil de consumo como a informação de consumo energético de um dispositivo para um dado intervalo de 15 minutos. Isto porque a framework já existente disponibiliza informação histórica de consumo com uma granularidade de 15 minutos. Quer isto dizer que à hora certa (por exemplo 14:00) temos acesso à informação sobre o consumo dos dispositivos durante os 15 minutos anteriores (a partir das 13:45).

Para cada dispositivo são enviados os perfis de consumo cronologicamente a partir da data indicada no ficheiro de configuração. É mantida a data do último perfil enviado para cada dispositivo. A pedido do cliente só são enviados 50 perfis de consumo de cada vez, e o dispositivo que tenha dados mais antigos ainda não enviados tem prioridade.

Quando existem perfis de consumo antigos por enviar a gateway cria JSON com 50 perfis de consumo e envia imediatamente, as vezes necessárias para enviar toda a informação em atraso. A maior parte do tempo, no entanto, a gateway terá todos os dados que estão disponíveis enviados para o endpoint do cliente. Nesta situação é interessante impedir que todas as gateways tentem enviar os perfis novos assim que estes ficam disponíveis, visto que isso quereria dizer que todas as gateways enviariam mensagens quase ao mesmo tempo. Como foi discutido em 4.1.1, este comportamento não é desejável. Por estas razões quando o bundle de cada gateway esgota o histórico de perfis de consumo para todos os dispositivos fica inactivo durante 15 minutos de forma a evitar que a rede seja inundada a cada 15 minutos.

Houve também algum esforço para generalizar este bundle de forma a poder ser configurado e usado em contextos semelhantes. Essencialmente parametrizar certos aspectos do funcionamento do bundle como o endpoint a utilizar para enviar os dados de consumo energético, o formato dos mesmos, o modo de autenticação utilizado, entre outros. Alguns destes pontos foram implementados ou parcialmente implementados, mas foi-me pedido para mudar o foco para outra área. A razão para isto foi o facto da framework OSGi no geral ainda estar a sofrer muitas mudanças e rapidamente trabalho que eu estaria a fazer teria de ser refeito devido a mudanças no resto da framework

4.2 Robot Framework

De momento não existe uma estrutura montada que permita uma verificação da qualidade do software de forma automática ou regular. De forma a melhorar e facilitar o desenvolvimento do produto. Uma bateria de testes que ponha à prova grande parte do sistema aumenta a confiança na funcionalidade do produto, a detecção atempada de erros e pela mesma razão poupa tempo e recursos à equipa de desenvolvimento.

Com este objectivo criei um conjunto de testes usando o paradigma da RobotFramework [4], uma framework de automação open source normalmente utilizada para testes automáticos. Nesta secção irei descrever os requisitos funcionais do projecto Redy no que toca a frameworks de automação de testes. Quero também fundamentar a escolha desta framework, e para tal vou explicar os básicos da sintaxe RobotFramework.

4.2.1 Requisitos

Actualmente cada membro da equipa testa as suas mudanças à medida que as implementa. Periodicamente, ou depois de uma mudança mais impactante são alocados alguns dias para testar todo o produto de forma a encontrar e resolver possíveis problemas recentemente introduzidos. Este método de desenvolvimento apresenta várias desvantagens.

Era necessária a montagem de um sistema que aliviasse este esforço periódico e facilitasse a detecção, o diagnóstico e resolução de problemas. Este seria o primeiro passo para, no futuro, se poder montar um sistema de Continuous Delivery.

Assim, era necessário escolher uma framework que fosse capaz de ser usada para testar todo o produto, se possível. Isto é interacções entre dispositivos, gateway, serviços da AWS e app. Como seria de esperar, os resultados devem ser apresentados de uma forma directa e sucinta, de forma a que o problema seja mais facilmente identificado. Por último seria também interessante que a criação de novos testes seja fácil de forma a promover uma maior cobertura.

4.2.2 Sintaxe

Neste último campo a sintaxe dos testes RobotFramework pode ser vantajosa. O seu uso das chamadas keywords pode facilitar muito a criação de novos testes. Vou explicar brevemente nesta secção as funcionalidades mais básicas e centrais da framework. Entrarei em maior detalhe no capítulo da implementação 5.1

Um teste RobotFramework simples é constituído por uma ou mais keywords. Neste exemplo vemos a definição de um teste que contem apenas uma keyword.

```
1 *** Test Cases ***
2 User Management Post Requests
3   Post Sample User
```

A definição de uma keyword pode existir numa biblioteca, ser implementada em Java ou Python ou ser composta por uma ou mais keywords. Aqui vemos a decomposição da keyword que compõe o teste em três keywords diferentes.

```
1 *** Keywords ***
2 Post Sample User
3     ${reply}= Post user      ${USER1}
4     Set Test Variable        ${USER_ID}    ${reply.json()["userId"]}
5     Check status code        ${reply}      200
```

Esta possibilidade de composição de keywords faz com que possamos ter na definição do teste keywords com níveis de abstracção muito altos. Keywords são também bastante flexíveis, podendo retornar valores e receber argumentos de várias formas.

4.2.3 Escolha

A utilidade de uma bateria de testes depende de muitos factores. Um dos mais importantes é a percentagem da funcionalidade coberta pelos testes. Não serve de muito um sistema passar um conjunto de testes se estes testes apenas cobrem uma pequena parte do sistema. Para um conjunto de testes automáticos ter alguma utilidade, tem de haver confiança que um sistema que passe todos os testes é um sistema estável e sem falhas.

A sintaxe do RobotFramework é vantajosa neste aspecto. Com o encadeamento certo de keywords é possível criar quase uma linguagem própria. Isto faz com que seja possível que até membros da equipa que não saibam programar consigam escrever novos testes.

A RobotFramework é open source e pode ser estendida por bibliotecas. Sendo a framework razoavelmente popular, algumas das suas bibliotecas são igualmente populares, especialmente as que facilitam o teste de interfaces web e mobile como a *AppiumLibrary* [22] para testes em aplicações iOS e Android, a *SeleniumLibrary* [23] que usa internamente a ferramenta *Selenium* para automatizar acções em browsers, entre outras. Com estas ferramentas é possível testar todos os aspectos do sistema Redy em separado e fazer testes end-to-end quando necessário.

A última vantagem a realçar desta framework é a forma como os resultados são apresentados. A framework gera uma página HTML com a informação importante imediatamente consultável. Esta página espelha a sintaxe dos próprios testes e encapsula as keywords mais específicas dentro de campos expansíveis das keywords mais genéricas. Quando os testes correm com sucesso todos os campos aparecem colapsados e é possível ter uma visão geral do teste corrido, visto que apenas estão visíveis os campos relativos às keywords mais genéricas. Quando o teste falha são expandidos os campos relativos à keyword que falha e às keywords por cima desta na pilha de execução. Assim é fácil e imediato inspeccionar a causa do problema porque temos toda a pilha de execução expandida e pronta para ser examinada.

No final foi escolhida a RobotFramework por ser já ligeiramente conhecida pela equipa, poder ser usada para testar todo o produto, ter uma sintaxe que pode vir a facilitar a criação de novos testes e a forma como apresenta resultados tornar muito fácil a detecção do problema quando um teste falha e gerar estatísticas interessantes quando o teste passa.

Capítulo 5

Implementação

Neste capítulo pretendo apresentar com detalhe as decisões que tomei durante o meu estágio, no que toca à implementação da solução. À semelhança do capítulo anterior, irei partir o meu trabalho em duas secções. A primeira parte incide sobre o bundle Java OSGi e a segunda incide sobre a bateria de testes RobotFramework desenvolvida.

5.1 Java OSGi

5.1.1 Dependências da Framework

Para o meu bundle funcionar, são necessários os serviços de três bundles já existentes na framework.

Device Access Service Responsável por acesso às informações actuais de um periférico como o seu estado, consumo instantâneo, leitura e escrita de propriedades e registo de notificações periódicas ou de mudança de uma determinada propriedade.

Device Event Notifier Service Bundle utilizado para receber notificações de entradas e saídas de periféricos.

Historic Data Service Responsável por guardar e disponibilizar histórico das propriedades de todos os devices.

5.1.2 Estrutura de classes

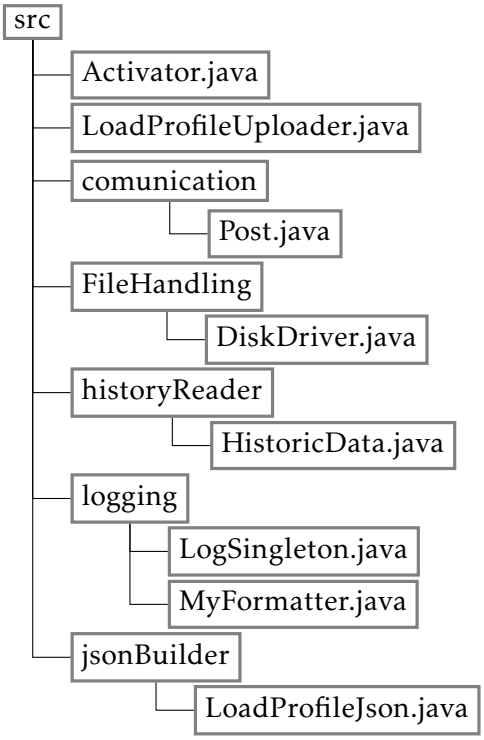

Figura 5.1: Estrutura de ficheiros do bundle OSGi

Activator.java Classe responsável pelo início de vida do bundle. Simplesmente espera que os serviços necessários se registem na framework e de seguida inicia o funcionamento deste bundle.

LoadProfileUploader.java Trata de inicializações. Inicia listeners subscritos no bundle "Device Event Notification Service"para cuidar da entrada e saída de periféricos.

HistoricData.java Classe responsável pela interface com o bundle "Historic Data Service". Extrai do bundle perfis de consumo de um determinado periférico e envia-os para a classe Post.java.

Post.java Classe responsável pela criação dos objectos JSON que representam os perfis de consumo e envio dos mesmos para o exterior.

logging Package responsável pelo log da aplicação. Sendo impossível testar o bundle na máquina onde se trabalha é de extrema importância que os logs da aplicação sejam legíveis e bem guardados.

LoadProfileJson.java Classe utilizada para simplificar a criação dos objectos JSON, visto que estes podem ficar um pouco complicados com várias datas, periféricos e devices.

5.1.3 Demonstração

Nesta secção vou expor as partes mais importantes do código da solução desenvolvida.

Como já foi dito, o inicio de vida do bundle dá-se na classe "Activator". Esta classe é responsável por aguardar que todos os serviços necessários estejam disponíveis antes deste bundle iniciar a sua actividade.

Quando o bundle é iniciado a função start(), linha 8, é chamada. Nesta função são criados ServiceTrackers e ServiceTrackerCustomizers para cada um dos três serviços necessários. No exemplo consta a operação para o serviço Historic Data Service, mas a mesma é idêntica para os restantes serviços.

O ServiceTracker, depois de criado e aberto nas linhas 19-21, é chamado pelo BundleContext sempre que o serviço que lhe foi atribuído bundle é adicionado, removido ou modificado. Os ServiceTrackerCustomizers especificam o comportamento desejado quando cada um destes eventos acontece.

Quando os três serviços estiverem funcionais uma thread nova é criada para a execução do bundle. Este comportamento faz parte do contracto OSGi.

Na figura 5.2 podemos ver a implementação do construtor da classe LoadProfileUploader e o método run(), os dois métodos chamados na classe Activator. No constructor o ficheiro de configuração é lido. Consoante a informação lida um objecto da classe Post é iniciado. A classe DiskDriver é usada para recuperar as datas já enviadas de cada periférico. Se nada for encontrado, um mapa vazio é criado. De seguida é instanciada a classe HistoricData, que vai ser responsável por todo o restante trabalho de ir buscar dados históricos ao Historic Data Service, tratar os mesmos, enviá-los utilizando a instância da classe Post criada e manter registo em disco de até que datas os dados de cada periférico foram enviados.

A função run(), necessária para implementar a interface Runnable, simplesmente chama a continuamente o método sendData() da classe HistoricData.

Este método, mostrado na figura 5.3, procura o periférico cujos últimos dados enviados sejam mais antigos. Verifica se os dados deste periférico foram sincronizados na actual janela de 15 minutos. No caso afirmativo a classe Post envia todos os perfis de consumo que tenha por enviar e o bundle fica inactivo durante 15 minutos. De seguida o método retorna para a linha 54 da figura 5.2 e é chamado de novo imediatamente. No caso negativo, o serviço Historic Data Service é utilizado para recuperar a informação de consumo do periférico em questão (o nome da variável que precisamos é "energy_aplus_inc", cada periférico tem muitas e diferentes variáveis), no período entre a sua última sincronização e o presente. Havendo informação para enviar, a mesma é enviada pela classe Post. Existe um limite imposto pelo cliente B2B de no máximo 50 perfis de consumo por cada mensagem enviada. Por isto não há aqui garantias que toda a informação enviada para Post é enviada para o endpoint do cliente, e por isto há a necessidade do método append(), linha 37, retornar a data do último perfil enviado. Esta data é então guardada no mapa e o método é chamado de novo. O periférico mais antigo no mapa é escolhido e o processo

repete-se.

Listagem 5.1: Excerto da classe Activator

```
1  public class Activator implements BundleActivator {
2
3      private BundleContext bc;
4      private DeviceAccessService dService;
5      private HistoricDataService hService;
6      private DeviceEventNotifierService denService;
7
8      public void start(BundleContext context) throws Exception {
9          this.bc = context;
10         ServiceTrackerCustomizer historicST = new ServiceTrackerCustomizer() {
11             @Override
12             public Object addingService(ServiceReference sr) {
13                 hService = (HistoricDataService) bc.getService(sr);
14                 System.out.println("HistoricDataService service added");
15                 launchIfReady();
16                 return hService;
17             }
18         };
19         new ServiceTracker( bc,
20               HistoricDataService.class.getName(),
21               historicST).open();
22     }
23
24     public void launchIfReady() {
25         if (dService != null && hService != null && denService != null) {
26
27             LoadProfileUploader lpu =
28                 new LoadProfileUploader(dService, hService, denService);
29
30             new Thread(lpu).start();
31         }
32     }
33 }
```

Listagem 5.2: Excerto da classe LoadProfileUploader

```
1  private Post              post;
2  private HistoricData      historicData;
3  private Map<String, Long> timestampMapDisk;
4
5  public LoadProfileUploader(
6      DeviceAccessService         dService,
7      HistoricDataService         hService,
8      DeviceEventNotifierService  denService) {
9
10   readFileProps();
11   initPost();
12   DiskDriver dd           = new DiskDriver();
13   this.timestampMapDisk   = dd.readLastSynchedTimestamps();
14
15   this.dService           = dService;
16   this.historicData       = new HistoricData(this.post, hService);
17   LOGGER.log(Level.INFO, "LoadProfileUploader" , "Historic Data Started");
18
19   DeviceEventListener deviceEventListener = new DeviceEventListener() {
20     @Override
21     public void deviceJoined(String device) {
22       deviceIn(device);
23     }
24     @Override
25     public void deviceLeft(String device) {
26       deviceOut(device);
27     }
28     @Override
29     public void deviceOnline(String device) {
30       deviceIn(device);
31     }
32     @Override
33     public void deviceOffline(String device) {
34       deviceOut(device);
35     }
36     @Override
37     public void deviceUpdated(String string) {
38     }
39   };
40
41   denService.register(deviceEventListener, "");
42   LOGGER.log( Level.INFO,
43         "LoadProfileUploader" ,
44         "DeviceEventListener Started and registered");
45 }
46
47 @Override
48 public void run() {
49   LOGGER.log( Level.INFO,
50         "LoadProfileUploader" ,
51         "LoadProfileUpdater Running");
52
53   while(true){
54     historicData.sendData();
55   }
56 }
```

Listagem 5.3: Excerto da classe HistoricData

```
1  private static final long FIFTEEN_MINUTS_MS = 900000;
2  private ConcurrentHashMap<String, Long> nodeTimestampMap;
3
4  public void sendData(){
5
6      long nowTS          = new Date().getTime();
7      long last15mins     = nowTS - (nowTS%FIFTEEN_MINUTS_MS);
8
9      String nodeToUpdate = chooseOldestNode();
10
11     if(nodeTimestampMap.get(nodeToUpdate) > last15mins){
12         LOGGER.log( Level.INFO,
13                     "HistoricData",
14                     "There is no new data. " +
15                     "Sending pending load profiles and sleeping");
16         this.post.forceSend();
17         try {
18             Thread.sleep(FIFTEEN_MINUTS_MS);
19         } catch (InterruptedException ex) {}
20         return;
21     }
22
23     List<HistoricProperty> histProps =
24                 getHistoricProperties( nodeToUpdate,
25                                        "energy_aplus_inc",
26                                        nodeTimestampMap.get(nodeToUpdate),
27                                        nowTS);
28
29     if(histProps == null || histProps.isEmpty()){
30         nodeTimestampMap.replace(nodeToUpdate, nowTS);
31     }else{
32         LOGGER.log( Level.INFO,
33                     "HistoricData" ,
34                     "Sending " + nodeToUpdate +" -> " +
35                     histProps.size() + " readings to Post");
36
37         long newTS = this.post.append(nodeToUpdate, histProps);
38
39         nodeTimestampMap.replace(nodeToUpdate, newTS );
40     }
41 }
```

5.1.4 Dificuldades de desenvolvimento

A principal dificuldade na elaboração deste bundle vem do facto de a única maneira de o testar é corre-lo numa gateway física, com periféricos conectados e dados reais. Isto e o facto da documentação da framework na altura ser muitas vezes insuficiente implicou muito tempo gasto em tentativa e erro. Encontrar e resolver um bug pode demorar mais que um dia porque cada mudança no código tem de ser compilada num ficheiro .jar, este tem de ser transferido para a gateway através de uma ligação SSH e a framework tem de ser reiniciada. Neste aspecto, sofri do mesmo problema que o resto da equipa na medida em que muito tempo foi gasto em testes e procura de problemas. Um deles foi o facto de já depois do bundle estar em produção em algumas gateways do lado do cliente este reportar que todas gateways estavam a não enviar alguns perfis de consumo. Não parecia haver nenhum padrão que ditasse quais são ignorados ou quais enviados. Eventualmente o problema foi descoberto. A framework disponibiliza os dados em janelas de 15 minutos, mas a informação só fica disponível no Historic Data Service algum tempo depois. Estamos a falar de alguns segundos a um minuto, o que era suficiente para o método getHistoricProperties() na linha 24 da figura 5.3 correr por exemplo às 12:00:01 e os dados referentes ao período 11:45 -> 12:00 não estarem ainda disponíveis.

5.2 RobotFramework

Estrutura de ficheiros da solução final de testes:

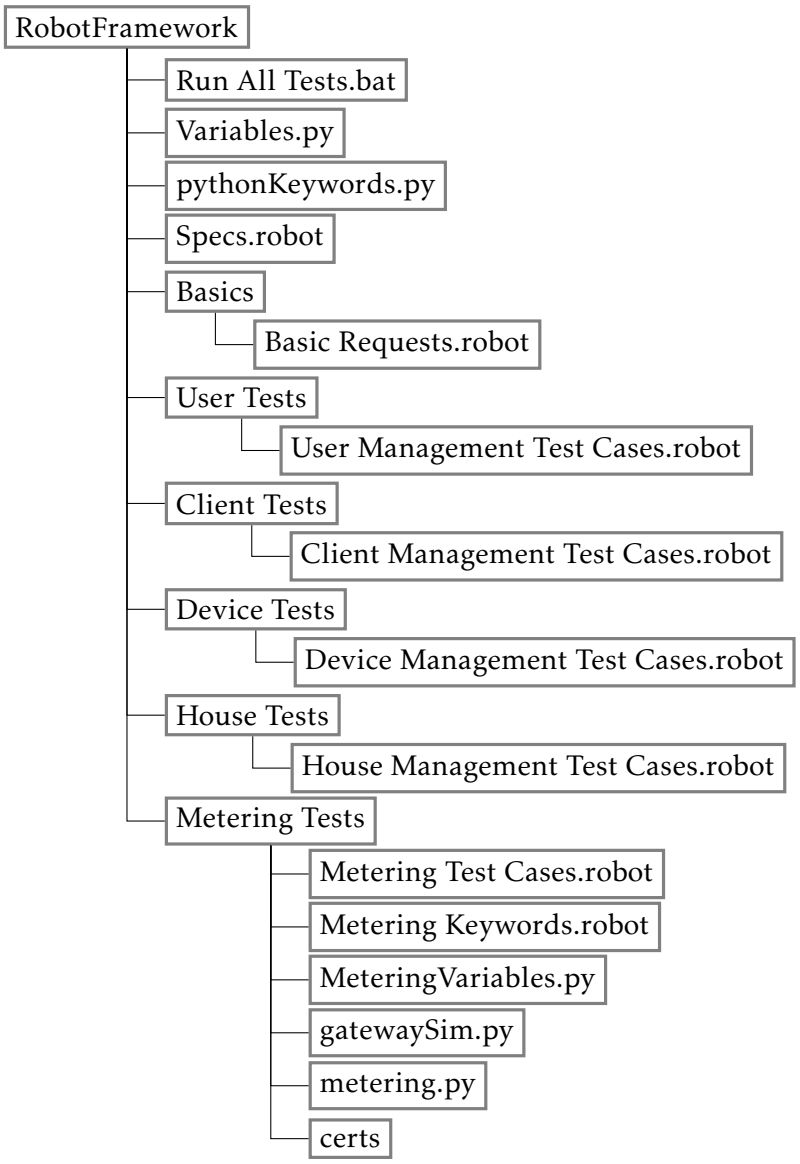

Figura 5.2: Estrutura de ficheiros dos testes RF

Run All Tests Ficheiro executável simples que corre os test cases em cada um dos ficheiros .robot

Variables.py Ficheiro com a declaração de todas as variáveis e dados usados nos testes

Python Keywords Implementação de keywords em Python, quando estas eram demasiado complexas para a sintaxe RobotFramework

Specs.robot Definição das keywords usadas nos testes. Uma camada de abstracção média entre os test cases e o conteúdo da pasta "Basics"

Basics Todos os requests à API que suporta o Redy estão definidas aqui

*** Tests** Definição dos test cases, uma pasta para cada API (Users, Clients, Houses, Devices). Quando o ficheiro executável é corrido, o output de cada teste é guardado na respectiva pasta.

Metering Tests Definição de testes de metering e código Python necessário para a simulação de uma gateway

Metering Test Cases.robot -> Definição dos test cases

Metering Keywords.robot -> Ficheiro auxiliar que implementa as keywords de alto nível em "Metering Test Cases.robot"

MeteringVariables.py -> Script Python com a definição de váriáveis necessárias apenas para os testes de metering

gatewaySim.py -> Script Python que abstracta toda a comunicação com a AWS IOT.

metering.py -> Script Python que implementa keywords complexas necessárias ao teste de metering.

certs -> Pasta que contem os certificados, chaves públicas e chaves privadas necessárias para a autenticação com a AWS IOT.

O output é gerado após a execução de cada ficheiro de teste na sua respectiva pasta. A execução do "Run All Tests.bat"vai mostrando o progresso e o resultado dos testes à medida que corre. Após a execução o resultado de cada teste é guardado na respectiva pasta.

5.2.1 Exemplo

Nesta secção vou mostrar parte do código que compõe um dos testes de metering. Aqui vemos a definição de alto nível do caso de teste.

Listagem 5.4: Excerto de Metering Test Cases.robot

```
*** Test Cases ***
Consumption Tarif 1
  With a simple tariff Consumption house
  When constant consumption is reported
  Then metering is corrrect for simple tariff
```

Vemos que este test case é composto apenas por três keywords. A implementação destas keywords encontram-se no ficheiro "Metering Keywords.robot".

Listagem 5.5: Excerto de Metering Test Keywords.robot

```
*** Keywords ***
With ${keyword}
  Run Keyword     ${keyword}
When ${keyword}
  Run Keyword     ${keyword}
Then ${keyword}
  Run Keyword     ${keyword}

a ${tariff_type} tariff ${house_type} house
  Log   House Type: ${house_type}
  Log   Tariff Type: ${tariff_type}
  Post House with tariff    ${house_type}   ${tariff_type}

constant consumption is reported
  Report constant consumption

metering is corrrect for ${tariff_type} tariff
  Log   ${tariff_type}
  Metering should match   ${tariff_type}
```

A implementação destas estão ou no ficheiro "Basic Requests.robot"onde estão todos os requests à APIGateway, ou estão definidas em Python, como o exemplo abaixo:

Listagem 5.6: Excerto de Metering Test metering.py

```
1   def Metering_should_match(tariff_type):
2
3     if(tariff_type != "simple"):
4       callKeyword("Fail", arg1 = tariff_type + not implemented)
5
6     reply = callKeyword_expect("Get Energy Chart",
7                                200,
8                                arg1 = houseId,
9                                arg2 = deviceId,
10                               arg3 = moduleId,
11                               arg4 = params)
12
13    acceptableError     = 0.00001
14    simpleExpectedCost  = 4.032
15    simpleExpectedValue = 0.96
16    cost                = getCost(reply)
17    value               = getValue(reply)
18
19    if( cost - simpleExpectedCost  )  > acceptableError):
20      callKeyword("Fail",
21                  arg1 = "Expecting " + simpleExpectedCost + ", got " + cost)
22
23    if( value - simpleExpectedValue ) > acceptableError):
24      callKeyword("Fail",
25                  arg1 = "Expecting " + simpleExpectedValue +
26                  ", got  "+ value)
```

Aqui podemos ver um exemplo de como uma keyword com espaços e parâmetros pode ser implementada em Python. Da prespectiva do developer não há nenhum passo intermédio entre a interpretação da linha 19 da figura 5.5 e a interpretação da função definida em 5.6.

Capítulo 6

Conclusões

Gostava agora de expor resumidamente os resultados e aprendizagens que resultaram da minha colaboração com a equipa de desenvolvimento do EDP Redy. Primeiro em termos de cada uma das minhas contribuições e por fim fazendo um balanço geral de toda a experiência que foi trabalhar no "mundo real".

6.1 Java OSGi

O bundle OSGi criado para o cliente B2B continua em funcionamento em todas as gateways do projecto. As necessidades do cliente não mudaram nesse tempo e apesar da framework OSGi ter sofrido bastantes alterações nada no bundle desenvolvido teve de ser adaptado. Estão desde que o meu estágio acabou, 400 gateways a correr nos parques de antenas da operadora.

Este número podia ter aumentado, pois existem ainda parques que não utilizam o serviço Redy e cujos consumos energéticos não são monitorizados. Durante o decorrer do projecto chegou-se à conclusão que o hardware do produto não é adequado e não aguenta suficientemente bem as condições a que está sujeito descritas em 4.1.1. Normalmente estão à volta de 10% da gateways offline, em parte por falhas nos routers e no sistema de comunicação e em parte por falhas do hardware Redy. Esforços foram feitos para remediar a situação, como fabricar uma versão mais robusta de alguns componentes, mas nenhuma provou ser suficiente. Este contracto não justifica uma mudança de rumo por parte da equipa do Redy e portanto não é previsível que o volume de negócio com este cliente neste projecto vá aumentar.

6.2 Robot Framework

O conjunto de testes que escrevi foi guardado, documentado e apresentado por mim à equipa antes do fim do meu estágio. Infelizmente este trabalho não foi continuado embora tenha sido alvo de alguma atenção por parte de dois colegas. Um testou a viabilidade de se usar a RobotFramework para testar a app e outro para testar em profundidade os serviços da AWS. Ambos chegaram à mesma conclusão: é uma maneira viável de se implementar testes automáticos, no entanto é preciso um investimento de tempo grande. Ainda mais porque para se tomar realmente partido da ferramenta era preciso ter um pipeline de versionamento, teste e deploy montado, e isso ainda implica mais mudanças e mais recursos. O trabalho que desenvolvi continua documentado e pronto a ser utilizado e expandido quando se considerar que o fazer é o uso mais eficiente dos recursos disponíveis. Até tal momento, o foco da equipa e de outros estágios depois do meu foi de construir features novas e crescer o produto.

6.3 Considerações finais

O meu estágio na EDP Comercial teve lugar há aproximadamente um ano. Durante esse tempo o produto e a equipa continuaram a evoluir. Não abandonando o Redy, a equipa começou a desenvolver um produto novo também relacionado com energy management, mas mais direccionado ao mass market. Este produto terá como objectivo a desagregação de cargas usando um processo chamado **NILM** - Non Intrusive Load Monitoring. Este processo permite que a partir apenas da monitorização do consumo geral de uma habitação se deduza que equipamentos são utilizados e os seus consumos individuais. Será uma alternativa muito mais barata e simples que o Redy, mas não traz qualquer capacidade de actuação sobre os equipamentos, apenas consegue dar informação um pouco mais detalhada sobre os consumos energéticos da casa. Outra diferença importante é que este dispositivo se liga directamente à Wifi do cliente, não precisando de uma gateway como intermediário.

Com esta mudança para ter um foco maior no mass market, a EDP Comercial espera duplicar ou triplicar o número actual de dispositivos a comunicar com a infraestrutura AWS até ao fim de 2021. Tal ambição traria problemas de escalabilidade com plataformas antigas, mas com os serviços da Amazon não se antecipam quaisquer problemas.

Se a EDP tiver sucesso em penetrar o mercado desta forma é provável que haja finalmente a necessidade e os recursos necessários para desenvolver um processo de desenvolvimento, teste e deploy mais eficiente e automático, pelo que o meu trabalho com Robot Framework pode ainda vir a ver uso no mundo real.

Este estágio preparou-me para a minha vida profissional de várias formas. O processo selecção foi independente da faculdade e idêntico ao do "mundo real", tive oportunidade de ver como uma equipa de desenvolvimento se organiza e como é gerida conforme o contexto em que está inserida e os resultados do passado. Tive de interagir e recolher

conhecimento de várias pessoas de várias áreas e até de outras equipas externas à EDP. Conheci a realidade que um projecto enfrenta tanto no seu dia a dia como ao longo de vários meses em que o mercado muda e o produto evolve com o mesmo. Tive a oportunidade de ver um bom líder de equipa em acção e de fazer parte de uma forte equipa.

Sinto que desempenhei o meu trabalho com competência e qualidade e atingi os objectivos que me foram propostos pela equipa de desenvolvimento do EDP Redy. No processo tive contacto com o desenvolvimento software num contexto de indústria e onde o meu trabalho tem implicações reais no mundo real. Sinto-me preparado para entrar no mercado de trabalho com confiança que o meu conhecimento, capacidade de aprendizagem e adaptabilidade acrescenta valor a qualquer equipa.

Bibliografia

[1] *Serviço EDP Redy*. URL: https://www.edp.pt/particulares/servicos/redy/.

[2] *Amazon Web Services*. URL: https://aws.amazon.com/.

[3] *OSGi*. URL: https://www.osgi.org/developer/what-is-osgi/.

[4] *RobotFramework*. URL: https://robotframework.org/.

[5] *Amazon home assistant features*. URL: https://www.amazon.com/b?node=17934672011.

[6] *Google home assistant features*. URL: https://support.google.com/googlenest/answer/9583920?hl=en&ref_topic=9831837.

[7] *Apple home assistant features*. URL: https://support.google.com/googlenest/answer/9583920?hl=en&ref_topic=9831837.

[8] *Zigbee*. URL: https://zigbeealliance.org/solution/zigbee/.

[9] *Samsung Smartthings*. URL: https://www.smartthings.com/.

[10] *Hubitat*. URL: https://hubitat.com/.

[11] *Xiamoi Smart Home*. URL: https://xiaomi-mi.com/mi-smart-home/.

[12] *Busybox*. URL: https://busybox.net/.

[13] *Continuous Integration, Delivery and Deployment*. URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[14] *Jenkins*. URL: https://www.jenkins.io/.

[15] *AWS CodePipeline*. URL: https://aws.amazon.com/codepipeline/.

[16] *AWS - IOT*. URL: https://aws.amazon.com/iot/.

[17] *AWS - API Gateway*. URL: https://aws.amazon.com/api-gateway/.

[18] *AWS - DinamoDB*. URL: https://aws.amazon.com/dynamodb//.

[19] *AWS - Lambda*. URL: https://aws.amazon.com/lambda/.

[20] *AWS - Cloudwatch*. URL: https://aws.amazon.com/cloudwatch/.

[21] *AWS - Cognito*. URL: https://aws.amazon.com/cognito/.

[22] *Appium library for RobotFramework*. URL: https://github.com/serhatbolsu/robotframework-appiumlibrary.

[23] *Selenium library for RobotFramework*. URL: https://github.com/robotframework/SeleniumLibrary/.