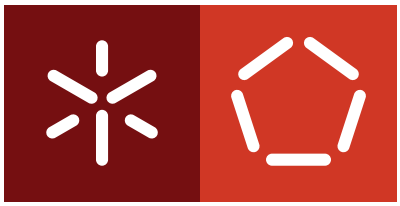
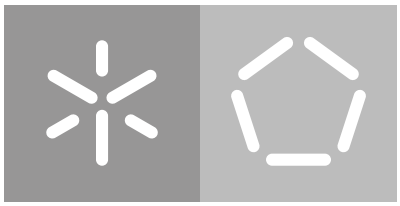


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Júlio Dinis Sá Peixoto

Gestão e controlo de dispositivos IoT através da interação com assistentes digitais

Outubro 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Júlio Dinis Sá Peixoto

Gestão e controlo de dispositivos IoT através da interação com assistentes digitais

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor António Nestor Ribeiro

Outubro 2019

AGRADECIMENTOS

Gostaria de expressar os meus sinceros agradecimentos para as pessoas que alguma forma contribuíram durante o meu percurso académico e na realização da presente dissertação.

Ao meu orientador, Professor António Nestor Ribeiro, pela oportunidade de realizar esta dissertação num tema do meu interesse, pelas ideias que foi sugerindo ao longo do seu desenvolvimento e ainda pelas inúmeras reuniões e tempo despendido a discutir detalhes da arquitetura e implementação do sistema resultante.

À equipa da *Google*, em especial ao Nick Felker, pela resposta rápida a todas as minhas dúvidas e às diversas discussões relacionadas com o *Google Assistant* e ferramentas disponibilizadas pela *Google* para o desenvolvimento das suas aplicações.

Aos meus pais e irmão, pelos princípios que me induziram, que fazem de mim a pessoa que sou hoje e por todo o apoio e suporte durante estes 5 anos.

À minha namorada, Rita, por estar constantemente a fazer-me sentir capaz de superar qualquer adversidade e sobretudo pela paciência nos momentos mais complicados deste último ano.

Aos meus amigos mais próximos por tornarem estes anos de vida académica inesquecíveis e por estarem sempre disponíveis para me ajudar.

À minha equipa na *Farfetch*, que me acompanhou durante este último ano, ouvindo as minhas mais diversas lamentações e faltas de motivação, nunca deixando que isto fosse razão para desistir.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração. Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

A crescente adopção de assistentes digitais faz com que os casos de uso para os quais estes são utilizados sejam cada vez mais abrangentes. Isto aliado ao facto dos dispositivos IoT estarem cada vez mais acessíveis, leva a que comece a ser comum os utilizadores controlarem diversos dos seus dispositivos através dos assistentes digitais.

O presente documento retrata a dissertação com o tema *Gestão e controlo de dispositivos IoT através a interação com assistentes digitais*. O principal foco desta passa pela investigação da gestão de múltiplos dispositivos de IoT, inseridos no mesmo ambiente e que possam ser geridos/controlados através de uma plataforma de interação por voz, neste caso um assistente digital.

Embora já existam soluções disponíveis para desenvolvimento de aplicações neste sentido, estas são muito recentes carecendo tanto ao nível do número de funcionalidades como da flexibilidade oferecida para sua utilização.

Esta investigação resulta numa proposta de arquitetura para aplicações deste domínio e da implementação de um sistema que permita fazer a gestão e controlo de um conjunto de dispositivos inteligentes inseridos no contexto de uma *smart home*. A arquitetura deste sistema e das aplicações que o constituem deverão possibilitar a inclusão de novas funcionalidades inexistentes nos assistentes digitais atuais e a implementação de algumas destas funcionalidades deverá ser apresentada como casos de estudo ao longo do presente documento.

Palavras-Chave: Arquitetura de aplicações; Assistentes digitais; Dispositivos IoT; Google Assistant.

ABSTRACT

The increasing adoption of digital assistants makes their use cases more widespread. This, coupled with the fact that IoT devices are becoming more and more accessible, increases the number of users that control many of their devices through digital assistants.

This document portrays the dissertation on *Managing and controlling IoT devices through digital assistants interaction*. The main focus is the investigation on multiple IoT devices management, taking in mind that these should be placed in the same environment and that can be managed/controlled through a voice interaction platform, in this case, a digital assistant.

Although there are already solutions available for building applications for digital assistants, they are very recent, lacking in both the number of features available and the flexibility offered for their use.

Thus, this research should result in an architecture proposal for applications within this domain and the implementation of a system that allows the management and control of a set of smart devices within the context of a smart home. The proposed architecture of this system and its applications should allow to include new features that do not exist in current digital assistants and the implementation of some of these features should be presented as case studies throughout this document.

Keywords: Applications architecture; Digital Assistants; IoT Devices; Google Assistant.

CONTEÚDO

1 INTRODUÇÃO 1
- 1.1 Contextualização 2
- 1.2 Motivação 3
- 1.3 Objetivos 4
- 1.4 Estrutura do documento 6

2 HISTÓRIA E REALIDADE ATUAL DOS ASSISTENTES DIGITAIS 8
- 2.1 História e relevância 8
  - 2.1.1 Internet of Things 10
- 2.2 Realidade atual 10
  - 2.2.1 Siri by Apple 10
  - 2.2.2 Alexa by Amazon 11
  - 2.2.3 Google Assistant by Google 12
  - 2.2.4 Cortana by Microsoft 13
  - 2.2.5 Bixby by Samsung 14
- 2.3 Âmbito da aplicação 14
- 2.4 Estado da arte 15

3 DESENVOLVIMENTO DE APLICAÇÕES PARA UM ASSISTENTE DIGITAL 18
- 3.1 Design da aplicação 18
  - 3.1.1 Seleção de casos de uso 18
  - 3.1.2 Criação de uma persona 19
  - 3.1.3 Escrita de diálogos 20
  - 3.1.4 Construção iterativa 22
- 3.2 Desenvolvimento 23
  - 3.2.1 Natural Language Processing 23
  - 3.2.2 Aplicações/Serviços 25
  - 3.2.3 Testes 26
- 3.3 Deploy 27

4 DESENVOLVIMENTO DE APLICAÇÕES PARA O GOOGLE ASSISTANT 28
- 4.1 Termos-chave 29
- 4.2 Tipos de ações 29
- 4.3 Fluxo dos pedidos 30
- 4.4 Interação dos utilizadores com ações 31
- 4.5 Descoberta e invocação de ações 33

4.5.1 Métodos de invocação 33
4.5.2 Métodos de descoberta 34
4.6 Dialogflow 34
4.6.1 Integração com ações 34
4.6.2 Desenvolvimento de um agente 35
4.6.3 Intenções 36
4.6.4 Entidades 37
4.7 Componente de lógica de negócio 38
4.8 Smart home actions 39
4.8.1 Intenções 40
4.8.2 Tipos de dispositivos 41
4.8.3 Tipos de funcionalidades 42
4.8.4 Request SYNC 43
4.8.5 Report State 43
5 APRESENTAÇÃO DO PROBLEMA 44
5.1 Desafios 46
6 ESPECIFICAÇÃO E MODELAÇÃO DO SISTEMA 48
6.1 Modelo de domínio 49
6.2 Selecção dos casos de uso 52
6.3 Criação de uma persona 53
6.4 Escrita dos diálogos 55
6.5 Especificação da API 57
6.5.1 Formato 58
6.5.2 Recursos 58
6.5.3 Rotas 60
6.5.4 Considerações 62
7 ARQUITETURA 64
7.1 Arquitetura geral do sistema 65
7.1.1 Infraestrutura Google 66
7.1.2 Firebase 67
7.1.3 Cloud Functions 67
7.1.4 Cloud Firestore 68
7.2 Arquitetura Serverless 69
7.3 Command-Query Responsability Segregation 72
7.3.1 Command-Query Separation 72
7.3.2 CQS vs CQRS 73
7.4 Arquitetura da Aplicação Smart home 77
7.4.1 Presentation layer 78

7.4.2 Application layer 79
7.4.3 Domain layer 80
7.4.4 Data layer 81
7.4.5 Infrastructure layer 81
7.5 Arquitetura da Aplicação Conversacional 81
7.5.1 Módulos NLP 82
7.5.2 Controladores 83
7.5.3 Message builders 83
8 IMPLEMENTAÇÃO 84
8.1 Tecnologias 84
8.1.1 JavaScript e Node.js 84
8.1.2 Programação assíncrona 85
8.1.3 Google JavaScript Style Guide 85
8.1.4 Cloud Functions 86
8.1.5 Firebase 86
8.2 Desenvolvimento das aplicações 87
8.2.1 Smart Home Action 87
8.2.2 Raspberry Pi 98
8.2.3 Conversational Action 99
8.2.4 Aplicação Smart Home Scheduler 107
8.3 Casos de estudo 109
8.3.1 Identificação de quais as funcionalidades de um determinado dispositivo 109
8.3.2 Agendamento de uma determinada operação 111
8.4 Desafios 112
8.4.1 Arquitetura Serverless 113
8.4.2 Command-Query Responsibility Segregation 113
8.4.3 Arquitetura da aplicação smart home 114
8.4.4 Arquitetura da aplicação conversacional 114
9 CONCLUSÃO E TRABALHO FUTURO 116
9.1 Experiência obtida 116
9.2 Trabalho futuro 117
9.3 Considerações finais 119

LISTA DE FIGURAS

Figura 1 História dos assistentes digitais até à atualidade 9
Figura 2 Funcionamento de aplicações para a Siri[4]. 11
Figura 3 Arquitetura de uma aplicação para a Alexa[2]. 12
Figura 4 Arquitetura de uma aplicação para o Google Assistant[15]. 13
Figura 5 Diagrama explicativo do funcionamento de aplicações para a Cortana[23]. 14
Figura 6 Exemplo do diálogo de um utilizador com uma aplicação. 21
Figura 7 Exemplo da tradução de uma mensagem em intenções e entidades. 24
Figura 8 Exemplo da utilização do Google Assistant em múltiplos dispositivos. [1] 28
Figura 9 Arquitetura e fluxo dos pedidos. [2] 31
Figura 10 Exemplo da utilização da ação GeekNum. 32
Figura 11 Exemplo da invocação da ação e respectiva resposta da mesma. 33
Figura 12 Funcionamento do Dialogflow perante uma fala de um utilizador. [14] 35
Figura 13 Exemplo da utilização de entidades do sistema. 37
Figura 14 Exemplo da utilização de entidades do programador. 38
Figura 15 Invocações entre Google Assistant, Dialogflow e Fulfillment. [3] 39
Figura 16 Alguns tipos de dispositivos disponíveis, retirado da documentação da Google Smart Home [15] 41
Figura 17 Alguns tipos de funcionalidades disponíveis, retirado da documentação da Google Smart Home [15] 42
Figura 18 Comparação entre o comportamento atual para uma query customizada (esquerda) com o pretendido depois de implementada a funcionalidade (direita). 45
Figura 19 Comparação entre o agendamento de comandos atual (esquerda) com o pretendido depois de implementada a funcionalidade (direita). 46
Figura 20 Funcionamento da *conversational action* e da *smart home action*. 47
Figura 21 Modelo de domínio 49
Figura 22 Logo do Dr. Smarthome 55
Figura 23 Diagrama do fluxo de diálogos 56

Figura 24	Diagrama da arquitetura geral do sistema.	65
Figura 25	Funcionamento das Cloud Functions [16]	68
Figura 26	Modelo de dados utilizado na Cloud Firestore [13]	69
Figura 27	Demonstração da aplicação do design pattern na prática.	73
Figura 28	Exemplo da perda de consistência no sistema.	76
Figura 29	Arquitetura da *Smart home Action*.	78
Figura 30	Arquitetura da *Conversational Action*.	82
Figura 31	Exemplo da criação de uma Cloud Function.	86
Figura 32	Dashboard das Cloud Functions.	87
Figura 33	Dashboard da Cloud Firestore.	87
Figura 34	Definição do nome da ação na plataforma *Actions on Google*.	88
Figura 35	Configuração do URL do *fulfillment* da ação na plataforma *Actions on Google*.	89
Figura 36	Configuração do *Account Linking* da ação na plataforma *Actions on Google*.	89
Figura 37	Excerto do código-fonte do *SmarthomeController*.	90
Figura 38	Excerto do código-fonte do *GetDevicesQueryHandler*.	91
Figura 39	Excerto do código-fonte do *GetDeviceStateQueryHandler*.	91
Figura 40	Excerto do código-fonte do *UpdateDeviceStateCommandHandler*.	92
Figura 41	Excerto do código-fonte do *OnOffService*.	93
Figura 42	Código-fonte do handler de pedidos POST no *UsersController*.	94
Figura 43	Excerto do código-fonte do *CreateUserCommandHandler*.	94
Figura 44	Excerto do código-fonte do *UsersWriteModelRepository*.	95
Figura 45	Diagrama da interação entre as aplicações smart home e Raspberry Pi.	98
Figura 46	Excerto do código-fonte da aplicação do Raspberry Pi.	98
Figura 47	Correspondência dos pins no Raspberry Pi (figura de cima) com o seu tipo (figura de baixo).	99
Figura 48	Configuração da ação de conversação na plataforma *Actions on Google*.	100
Figura 49	Exemplo de algumas das intenções disponíveis no Dialogflow.	101
Figura 50	Exemplo de *trainig phrases* para a intenção.	102
Figura 51	Exemplo de *Action and parameters* para a intenção.	103
Figura 52	Exemplo da definição do URL do *fulfillment*.	103
Figura 53	Exemplo das entidades criadas para utilização nas intenções.	104
Figura 54	Exemplo da definição da entidade *command_on_off*.	104
Figura 55	Exemplo da definição da entidade *device*.	105
Figura 56	Excerto do código-fonte de OnOffNlp.	106

Figura 57 Excerto do código-fonte de OnOffMessageBuilder. 106
Figura 58 Diagrama da integração das aplicações: Conversational Action, Scheduler Application e Smart home Action. 108
Figura 59 Diagrama de sequência da identificação de funcionalidades de um dispositivo. 110
Figura 60 Demonstração da identificação de funcionalidades de um dispositivo. 111
Figura 61 Diagrama de sequência do agendamento de uma determinada operação. 112
Figura 62 Demonstração do agendamento de uma operação (à esquerda) e notificação resultante quando a operação é realizada (à direita). 112

1

INTRODUÇÃO

O crescimento exponencial do mercado de smartphones e dispositivos conectados, com as suas respetivas aplicações, fez com que se tornassem num dos principais meios de comunicação e impusessem novas dinâmicas quanto à forma de interagir, pesquisar informação e realizar as mais diversas tarefas diárias. É neste contexto que começam também a surgir diversos assistentes digitais - aplicações com uma user-interface distinta já que a sua interação passa a ser realizada por voz - que vêm em auxílio do utilizador para realizar tarefas normalmente associadas às funções de um assistente pessoal.

Desde o aparecimento dos assistentes digitais no mercado dos smartphones, em 2011 quando a Apple introduziu a Siri[1], não passou muito tempo até que todos os smartphones e até computadores disponibilizassem o seu próprio assistente, permitindo aos utilizadores tirar partido de algumas das aplicações já existentes nestes dispositivos através da voz.

Em 2014 a Amazon lança o Echo, uma coluna inteligente cujo objetivo passa pela utilização do seu assistente digital, a Alexa[2]. O Echo introduz uma dinâmica diferente, uma vez que passamos a ter um dispositivo dedicado exclusivamente para a interação com um assistente digital e através do qual passa a ser possível realizar tarefas simples (consulta do tempo, lembretes, alarmes, etc) e até tarefas mais complexas como o controlo de diversos dispositivos inteligentes inseridos no mesmo ambiente. Um exemplo deste último seria ligar/desligar as luzes de uma determinada divisão da casa com um simples comando por voz. Além disto, permitiria ainda instalar aplicações desenvolvidas por terceiros, ficando assim com um dispositivo mais flexível e abrangente das diferentes necessidades de cada utilizador. O sucesso do Echo/Alexa fez com que começassem a surgir diversos concorrentes (Google Home, e mais recentemente o Apple HomePod), com cada vez aplicações mais distintas e completas e ainda novas frentes de investigação sobre o tema[3].

1 https://www.apple.com/siri/
2 https://developer.amazon.com/alexa

Os utilizadores estão a aderir em massa a estes dispositivos e é esperado que em 2021 cerca de 1.8 mil milhões de utilizadores façam uso de assistentes digitais regularmente[3], não só devido ao desejo por parte dos utilizadores de uma forma inovadora de interação com aplicações, neste caso através da utilização da voz, mas também devido à contínua evolução de inteligência artificial que permite o reconhecimento e tradução dos pedidos dos utilizadores, melhorando a experiência destes com os assistentes. O alcance de objetivos por meio da tecnologia poderá passar a seguir um padrão de interação por voz, que aliado ao crescimento exponencial da adesão dos utilizadores à utilização de assistentes digitais vem justificar a análise do uso destes para o efeito.

1.1 Contextualização

Neste momento já todas as grandes empresas de tecnologia têm o seu próprio assistente digital e aproveitam qualquer oportunidade para induzir os utilizadores a usarem-no, tirando partido das mais diversas funcionalidades neste embebidas. Uma grande diversidade de funcionalidades é um fator muito importante para o sucesso de um assistente digital já que todo o propósito deste é ter um propósito idêntico ao que um assistente pessoal teria - uma pessoa cujas habilidades lhe permitem fazer inúmeras tarefas para auxílio de outra.

A diversidade de funcionalidades presentes está diretamente relacionada com o número de aplicações que o assistente digital tem para oferecer. Se o desenvolvimento de aplicações ficar limitado à empresa que o mantém, o número de funcionalidades será, certamente, comprometido. Um exemplo real disto é o assistente digital Bixby que, por enquanto, oferece apenas aplicações desenvolvidas pela própria Samsung, limitando o número de funcionalidades oferecidas aos utilizadores. Este é um fator crítico que seguramente justifica a diferença significativa ao nível de utilizadores, quando comparado com as restantes alternativas.

O desenvolvimento de aplicações por parte de terceiros para os assistentes digitais é uma mais-valia para estes, já que permitem que tanto empresas como a própria comunidade trabalhem para o mesmo fim.

O desenvolvimento de aplicações para os assistentes digitais é uma prática muito recente e as funcionalidades disponíveis para os utilizadores são muito limitadas. Atualmente os utilizadores não podem perguntar ao Google Assistant quais as funcionalidades de um determinado dispositivo que podem controlar através da aplicação. Embora aparente ser algo trivial, em situações onde o número de dispositivos que o utilizador pode controlar

3 https://www.tractica.com/newsroom/press-releases/the-virtual-digital-assistant-market-will-reach-15-8-billion-worldwide-by-2021/

seja elevado pode tornar-se complicado distinguir quais as funcionalidades que cada um dos dispositivos dispõe. Deste modo, a procura de soluções para resolver e acrescentar algumas destas funcionalidades em falta fará também parte dos problemas a enfrentar no âmbito da presente dissertação.

Enquanto que já praticamente todos os assistentes existentes oferecem soluções de desenvolvimento de aplicações por parte de terceiros, ainda estão presentes algumas limitações que impedem os developers[4] de conseguir um resultado final completamente flexível para os utilizadores. Para além disto, toda a ideia de trazer a interação com os assistentes por voz é comprometida por muitas funcionalidades para as quais esta interação não é possível. Um exemplo simples disto são as *Google Routines* que permitem aos utilizadores agendar ações sobre os diversos dispositivos inteligentes das suas casas através de uma interface gráfica, no entanto não é oferecida a possibilidade do utilizador fazer o mesmo através da voz. Qualquer comando por voz para controlar um dispositivo através do Google Assistant é definido pela própria Google através de invocações padronizadas. Enquanto que isto é ideal para a maior parte das situações, já que a forma como os utilizadores invocam comandos acaba por seguir um padrão, no caso particular em que um utilizador pretenda utilizar uma frase customizada para realizar uma determinada ação sob um dispositivo inteligente, tal não é possível já que está previamente definido o que os utilizadores podem ou não dizer.

Por fim, o desenvolvimento de aplicações para assistentes digitais é ainda uma prática muito recente e como tal não existem convenções e/ou padrões arquiteturais bem definidos, o que leva a que as aplicações existentes sigam normas muito distintas entre si. É ainda de notar que grande parte das aplicações disponíveis não são *open-source*, abertas e disponíveis para a comunidade, mas sim desenvolvidas e mantidas por empresas. Neste sentido, a sistematização de padrões arquiteturais para aplicações deste domínio deverá também ser um dos temas abordados na presente dissertação.

1.2 MOTIVAÇÃO

O constante surgimento e desenvolvimento de dispositivos de IoT - *Internet of Things* - evidente pelo aparecimento de dispositivos inteligentes que, ao contrário dos dispositivos tradicionais, passam a disponibilizar uma ligação à internet, auxiliou o aparecimento de smart-speakers[5] - dispositivos físicos cujo principal propósito é a interação com assistentes digitais.

4 Ao longo do documento será utilizado o termo *developer* para designar não só o programador, mas também alguém que extravasa esse contexto definindo também a arquitetura da aplicação

5 https://www.iotworldtoday.com/files/2019/10/IHS-Markit-IoT-Device-Smart-Speaker-1.pdf

Segundo um estudo realizado no início deste ano pela Microsoft[9], atualmente cerca de 35% dos utilizadores fazem uso de assistentes digitais através destas smart-speakers e esta percentagem tende a aumentar. Enquanto que funcionalidades como pedir factos interessantes ou direções são das mais utilizadas com 68% e 65%, respetivamente, mais de metade dos utilizadores admite fazer uso das smart-speakers para gerir os dispositivos inteligentes presentes nas suas casas, seja para efeitos de iluminação ou até controlar sistemas de videovigilância.

As smart-speakers e respetivo assistente digital começam a ser vistas como uma plataforma centralizada onde os utilizadores podem controlar os seus diversos dispositivos de IoT[1]. Passa a ser possível, por exemplo, um utilizador pedir ao Google Assistant para ligar o aquecimento de sua casa quando a temperatura está demasiado baixa. Notemos que nesta situação para além de ser necessário o assistente digital reconhecer e entender o pedido do utilizador terá ainda de recolher periodicamente o valor da temperatura a partir de outros dispositivos conectados. Neste caso seria através de sensores de temperatura e assim que esta estivesse baixa, ligar efetivamente o aquecimento para satisfazer o pedido do utilizador.

O controlo de todos os dispositivos inteligentes presentes na sua casa está ao alcance da interação por voz com uma coluna inteligente. A tendência será ter várias smart-speakers espalhadas pela casa dos utilizadores para uma maior comodidade e interação mais pessoal com os assistentes digitais. Para satisfazer estas necessidades dos utilizadores, as aplicações para controlo de dispositivos conectados a partir destes assistentes e a sua investigação, serão cada vez mais um tema recorrente[5], numa constante procura em volta de metodologias, arquiteturas e soluções que possam agilizar a sua construção e desenvolvimento.

1.3 Objetivos

Os objetivos desta dissertação passam por conseguir uma solução para controlo de dispositivos IoT através de um dos diversos assistentes digitais existentes, isto é, conseguir que os utilizadores controlem os dispositivos presentes nas suas casas através de uma plataforma cuja interação é realizada por voz. Para ir ao encontro a este objetivo será necessário explorar a construção de aplicações gerais para estes assistentes e só posteriormente depositar o conhecimento adquirido com estas à construção de aplicações para interação com dispositivos inteligentes.

A solução deverá ser capaz de representar uma plataforma para gestão de múltiplos dispositivos inseridos no mesmo ambiente através da consulta e controlo do seu estado, sendo que a interface para submissão destes deverá ser um assistente digital. Esta solução

deverá procurar ser independente do assistente da qual fará uso para poder facilmente ser utilizada por qualquer um dos assistentes digitais existentes, se assim for a vontade do utilizador. Além disso, deverá ainda apresentar uma arquitetura escalável e configurável para que o número de dispositivos a gerir possa aumentar sem trazer transtornos e mudanças arquiteturais significativas.

Veremos adiante que algumas plataformas para desenvolvimento de aplicações para os assistentes digitais já auxiliam o desenvolvimento de aplicações direcionadas para a interação com dispositivos IoT, no entanto, as soluções oferecidas limitam muito a flexibilidade do programador no que diz respeito às interações dos utilizadores com os dispositivos para realizar determinados comandos. A solução final deverá permitir contornar esta falta de flexibilidade imposta pelas soluções anteriores, permitindo aos programadores definir os tipos de interação dos utilizadores para realizar determinados comandos, isto é, se um utilizador pretender dizer *Hello world* para ligar todos os dispositivos de iluminação presentes na sua casa, não deverão ser impostas quaisquer restrições para o efeito.

Com base nisto, apresentam-se de seguida, devidamente delineados os objetivos da presente dissertação:

- Investigar as diferentes plataformas oferecidas para desenvolvimento de aplicações para assistentes digitais, comparando-as sob a perspetiva de um programador;
- Apresentação de uma arquitetura base que cumpra com requisitos de escalabilidade, flexibilidade e manutenção para estas aplicações;
- Desenvolvimento de uma solução que permita os utilizadores controlar os seus dispositivos através de um assistente digital com base numa das plataformas anteriores;
- Apresentação de *case-studies* que demonstrem a possibilidade de contornar as limitações atualmente impostas pelas plataformas de construção de aplicações deste tipo.

Estes objetivos devem traduzir-se em não uma mas várias aplicações, sendo que o propósito e necessidade de cada uma destas é explicado com maior detalhe ao longo do presente documento. Identificados os problemas a resolver, resta enumerar as diferentes funcionalidades que as diferentes aplicações deverão apresentar.

- **Aplicação smart home**

  Aplicação responsável pela gestão de informação e estado dos dispositivos inteligentes dos utilizadores.

  - Identificação de uma arquitetura escalável para este tipo de aplicações;

  - Gestão e controlo de diversos dispositivos e funcionalidades seguindo os padrões recomendados pela Google (autenticação, *report state*, *homegraph*, etc);
  - Desenvolvimento de APIs para gestão de informação de utilizadores e dos seus dispositivos inteligentes;
  - Garantir que as alterações no estado dos dispositivos seja refletida em periféricos conectados a um Raspberry Pi;

- **Aplicação para Raspberry Pi**

  Aplicação para controlo dos dispositivos físicos, isto é, operações diretas sobre os periféricos conectados a um Raspberry Pi - por exemplo ligar e desligar um LED.

  - Visível para conexões exteriores à rede local onde se encontra;
  - Invocação de comandos sobre múltiplos periféricos conectados que simulam dispositivos inteligentes;

- **Aplicação conversacional**

  Aplicação responsável por, através de um módulo de *Natural Language Processing*, garantir flexibilidade nas invocações/mensagens dos utilizadores e a possibilidade de adição de novas funcionalidades ao assistente digital.

  - Gestão e controlo de diversos dispositivos não só com as invocações padrão já definidas mas também possibilitando a utilização de invocações personalizadas;
  - Obtenção de informação sobre as funcionalidades que os dispositivos de um determinado utilizador oferecem;
  - Agendamento de comandos sob dispositivos - por exemplo ligar um dispositivo a uma determinada hora do dia.

1.4 Estrutura do documento

O tema da dissertação foi introduzido com a contextualização, motivação e objetivos da mesma. Esta introdução será completada no capítulo seguinte onde, muito sucintamente, será descrita a história do progresso e desenvolvimento dos assistentes digitais e ainda da relevância e impacto que estes têm quando utilizados no domínio de IoT, isto é, para controlo de dispositivos inteligentes. Neste capítulo é ainda apresentada uma vista geral dos assistentes digitais existentes, sob a perspetiva de um programador, isto é, segundo as funcionalidades que as plataformas oferecem para construção de aplicações para estes assistentes.

Os dois capítulos que se seguem descrevem, detalhadamente, a construção de aplicações para um assistente digital, primeiramente de um modo geral e de seguida para o caso específico do Google Assistant. No primeiro são apresentadas diretrizes para o design destas aplicações, seguindo-se do processo de desenvolvimento que deve ser realizado para construção das mesmas e ainda fazendo referência aos diversos módulos que, regra geral, as compõem. No segundo são identificados os conceitos, metodologias, tecnologias e ainda ferramentas específicas que devem ser utilizadas na construção de aplicações para o Google Assistant. Por fim é apresentada uma secção que referencia a construção destas para utilização no contexto de uma smart home, isto é, para controlo e supervisão de dispositivos inteligentes inseridos no mesmo local.

No quinto capítulo é apresentado, com mais detalhe, o problema que se pretende solucionar com a presente dissertação. É neste capítulo que são enumeradas as aplicações a desenvolver em conjunto com uma vista geral da interação destas aplicações e das funcionalidades que devem dispor. Com o problema devidamente apresentado, o sexto capítulo contém a especificação e modelação do sistema a desenvolver. Neste capítulo são selecionados os casos de uso, assim como a criação de uma persona para a aplicação e a escrita do fluxo de diálogos que estará disponível através da mesma. A especificação do sistema termina com a apresentação da API a desenvolver.

No sétimo capítulo poderemos encontrar as propostas de arquiteturas para aplicações dentro do domínio dos assistentes digitais e da gestão de dispositivos inteligentes. Este capítulo começa por apresentar uma arquitetura geral do sistema, com especial referência para as tecnologias na *cloud* utilizadas para garantir que o mesmo seja viável. De seguida são abordados alguns padrões arquiteturais cujo foco está na escalabilidade, sendo que o capítulo termina com a apresentação das arquiteturas resultantes, onde estes padrões arquiteturais estão refletidos.

Evidenciada a arquitetura a desenvolver, no oitavo capítulo é explicado, sucintamente, o desenvolvimento das aplicações necessárias, colocando na prática tanto os conceitos abordados no capítulo da especificação e modelação do sistema como a arquitetura apresentada no capítulo anterior. Neste capítulo de implementação são referidas as diferentes tecnologias utilizadas e as razões que levaram à escolha de cada uma destas. Por fim, são abordados diferentes casos de uso onde são relatados os desafios e dificuldades que surgiram ao implementar as arquiteturas anteriores.

O documento termina com uma conclusão, onde é mencionada a experiência obtida durante a realização da dissertação e feita referência ao trabalho futuro que poderá ser realizado nesta vertente, relembrando que é um tema muito recente.

2

HISTÓRIA E REALIDADE ATUAL DOS ASSISTENTES DIGITAIS

Atualmente é possível realizar-se as mais diversas tarefas através da tecnologia. Neste contexto surge a necessidade de existência de uma plataforma comum para resolução destas tarefas e ainda, se possível, capaz de as automatizar ao máximo.

Os assistentes digitais auxiliam neste problema na medida que, para além de servirem como uma plataforma comum para as tarefas dos utilizadores ainda são capazes de interagir com estes através da principal forma de interação humana - a voz. A interação por voz faz com que os utilizadores se interessem verdadeiramente pelos assistentes digitais já que, se até então não estava ao alcance de todos ter um assistente pessoal para o qual podiam fazer pedidos através da voz, agora qualquer pessoa pode usufruir de tal.

2.1 HISTÓRIA E RELEVÂNCIA

Antes de abordarmos melhor os assistentes pessoais digitais podemos explorar um pouco da sua história, que começa numa altura em que a tecnologia era menos evoluída. O percurso até aos assistentes digitais como os conhecemos atualmente começa numa época em que a evolução da tecnologia era consideravelmente mais lenta do que na atualidade.

Os primeiros avanços dos assistentes digitais foram em 1961 quando a IBM introduz o IBM Shoebox[1], na altura uma ferramenta de reconhecimento de voz capaz de reconhecer umas incríveis 16 palavras e 9 dígitos. Quase 10 anos mais tarde, em 1971, a Universidade de Carnegie Mellon no decorrer de um programa de investigação de *Speech Understanding* desenvolve o Harpy[2], um sistema capaz de reconhecer cerca de 1000 palavras e frases completas.

1 https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html
2 https://www.totalvoicetech.com/a-brief-history-of-voice-recognition-technology/

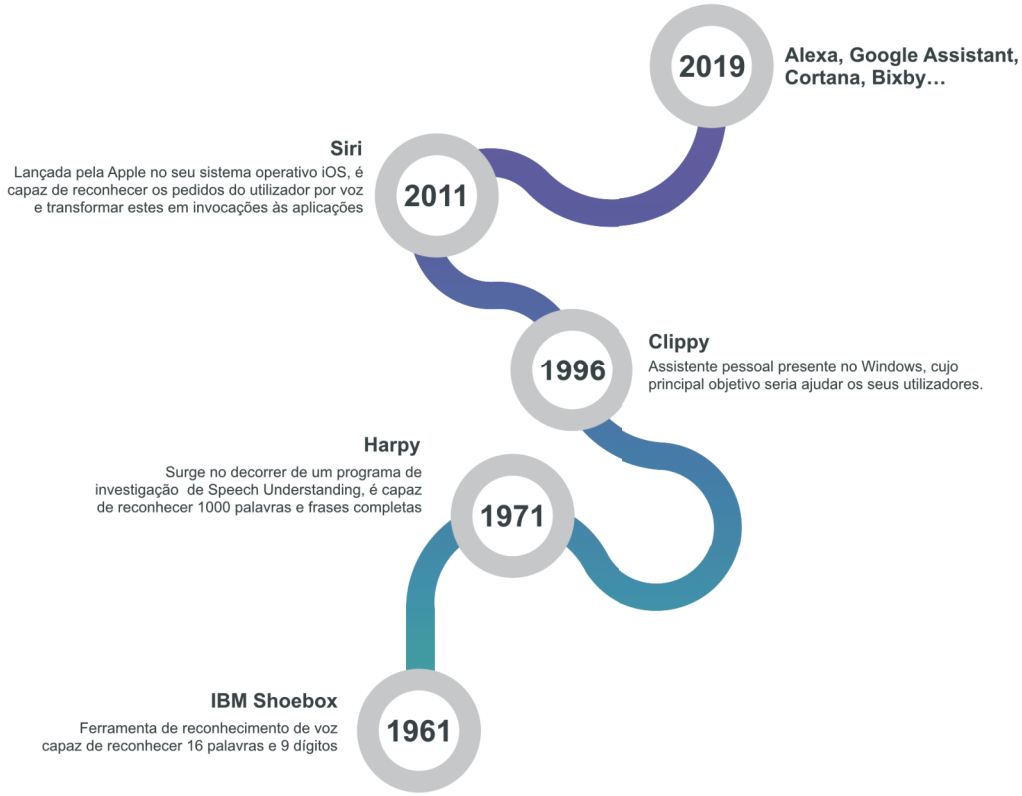


Figura 1: História dos assistentes digitais até à atualidade

Em 1996 a Microsoft introduz ao mundo o Clippy[3] que, embora não permitindo reconhecimento de voz tinha como objetivo servir como um assistente pessoal dos utilizadores, do que continua a ser o sistema operativo mais utilizado no mundo. É precisamente neste ano que os utilizadores comuns passam a ter a possibilidade de interagir com um assistente pessoal virtual. No entanto, o Clippy não foi um sucesso entre os utilizadores já que a Microsoft acabou por receber muitas queixas considerando-o intrusivo por este aparecer automaticamente e não apenas quando requisitado pelos utilizadores.

Mais recentemente, em 2011, a Apple lança a Siri[4], na altura o assistente digital mais completo, capaz de reconhecer os pedidos do utilizador por voz e satisfazer as suas necessidades através de invocações às aplicações contidas no smartphone. Posteriormente as grandes empresas tecnológicas começam a lançar as suas soluções: a Microsoft com a Cortana; a Google inicialmente com o Google Now que entretanto passaria a chamar-se Google Assistant; e a Amazon com a Alexa.

3 https://www.artsy.net/article/artsy-editorial-life-death-microsoft-clippy-paper-clip-loved-hate
4 https://www.cheatsheet.com/gear-style/how-old-is-siri.html/

Na secção seguinte abordaremos individualmente cada um destes assistentes, comparando-os sob a perspetiva de desenvolvimento de aplicações para os mesmos.

2.1.1 *Internet of Things*

Nesta altura é já possível encontrar pelo menos um assistente digital quer seja no computador, smartphone e até mesmo noutros smart devices, tipo smartwatches, speakers, carros ou outros. Estes assistentes surgem com o intuito de satisfazer as necessidades tecnológicas dos utilizadores que a cada ano que passa crescem tanto em número como em complexidade. Para muitos utilizadores o contexto de Internet of Things é onde estes assistentes fazem mais sentido já que permitem o controlo de diversos dispositivos conectados através de um canal único e interativo. Estes permitem controlar os diversos dispositivos inteligentes que têm em sua casa - nada melhor que poder controlar e automatizar diversos dispositivos não só através de uma plataforma única como através do mecanismo de interação de preferência dos humanos, a voz.

Facilmente conseguimos imaginar um futuro com a predominância da utilização da voz para realizar as mais diversas tarefas diárias dos utilizadores, tudo isto será possível através do desenvolvimento de um ou até múltiplos assistentes digitais que acompanharão os utilizadores tanto na realização das suas tarefas como também aconselhando-os nas decisões que tiverem de tomar.

2.2 REALIDADE ATUAL

Neste momento muitas das grandes empresas tecnológicas têm uma solução deste género para oferecer, é justo concluir que já reconheceram a importância e impacto que os assistentes digitais têm e ainda mais a que terão no futuro. Os assistentes digitais que se seguem serão comparados do ponto de vista de um programador, não no que diz respeito a quantidade e complexidade das aplicações que oferecem mas sim de possibilidades e diferentes alternativas para a construção destas aplicações.

2.2.1 *Siri by Apple*

A Apple produz software muito característico, em que regra geral é fechado aos seus produtos, não podendo ser utilizado pela concorrência. O mesmo acontece com o seu assistente digital, a Siri, estando apenas disponível para utilização nos sistemas operativos da Apple: macOS, iOS e watchOS.

As aplicações para a Siri focam-se em extensões para aplicações já existentes, deixando de fazer sentido a construção de uma aplicação de raiz para uso exclusivo com o assis-

tente. A Apple oferece assim aos programadores um conjunto de diferentes alternativas que permitem adicionar a invocação, através da Siri, de funcionalidades de uma aplicação já existente - SiriKit[4]. No contexto de smart home, isto é para controlo de dispositivos inteligentes, é oferecida exatamente a mesma alternativa, possibilitando a invocação da Siri através da sua speaker física, o Homepod, sendo que esta envia os pedidos recebidos para o smartphone do utilizador reutilizando a estratégia anterior.

A Figura 2 explica, de um modo geral, o funcionamento das aplicações para a Siri - *Intents App Extension* - cujo objetivo final é tirar proveito de funcionalidades de uma aplicação - *Your App* - já existente. As aplicações para a Siri definem os pedidos dos utilizadores que podem receber, considerados *Intents*, e traduzem estes em ações nas aplicações existentes no seu smartphone.

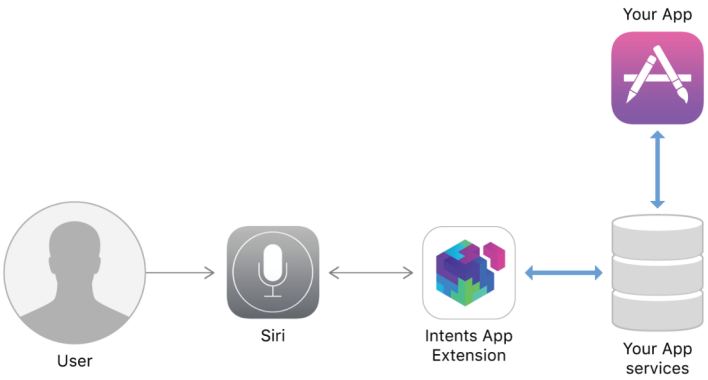


Figura 2: Funcionamento de aplicações para a Siri[4].

O desenvolvimento de aplicações no ecossistema da Apple tem uma filosofia de desenvolvimento muito característica, exigindo a utilização de metodologias e ferramentas próprias para o efeito - XCode, Swift, etc.

2.2.2 *Alexa by Amazon*

A Amazon oferece o Alexa Skills Kit[2], um conjunto de ferramentas, documentação e exemplos para auxiliar os programadores na construção de aplicações para a Alexa, o seu assistente digital. Assim como alguns dos seus concorrentes, a Amazon oferece integração com o seu serviço de cloud, o que permite acelerar todo o processo de desenvolvimento, em particular, da disponibilização da aplicação para ser utilizada por pessoas reais - deploy da aplicação.

Este kit distingue-se positivamente dos restantes já que disponibiliza APIs para diferentes tipos de aplicações, é exemplo disso a *Video Skill API* para construção de aplicações que

usam a reprodução de vídeo; *Music Skill API* para músicas; *List Skill API* para listagens, to-do ou shopping lists; *Smart Home Skill API* para diversos tipos de interação com smart devices, entre outras. Estas APIs incluem já todas as interações por voz do utilizador com a aplicação, o que simplifica bastante o processo de idealização e construção do módulo de *Natural Language Processing* (NLP).

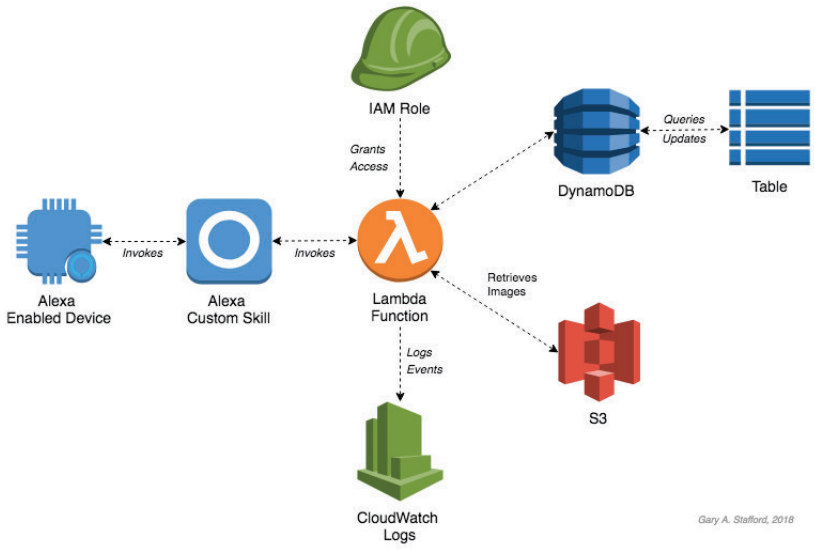


Figura 3: Arquitetura de uma aplicação para a Alexa[2].

A alternativa da Amazon acaba por ser tentadora já que oferece todo um ecossistema de ferramentas que facilita muito o desenvolvimento, deploy e manutenção de aplicações para a Alexa - Lambda Function, Amazon S3, Amazon DynamoDB.

2.2.3 *Google Assistant by Google*

A Google, assim como a Amazon, oferece todo um ecossistema de ferramentas para desenvolver aplicações para o seu assistente digital, o Google Assistant. Neste ecossistema podemos contar com o Dialogflow[5], para definição da interação/conversação com os utilizadores, Firebase[6] para host do serviço na cloud e como base de dados real-time e ainda o Actions on Google[15], ferramenta responsável pela gestão das aplicações (actions) para o assistente.

5 http://dialogflow.com/
6 http://firebase.google.com/

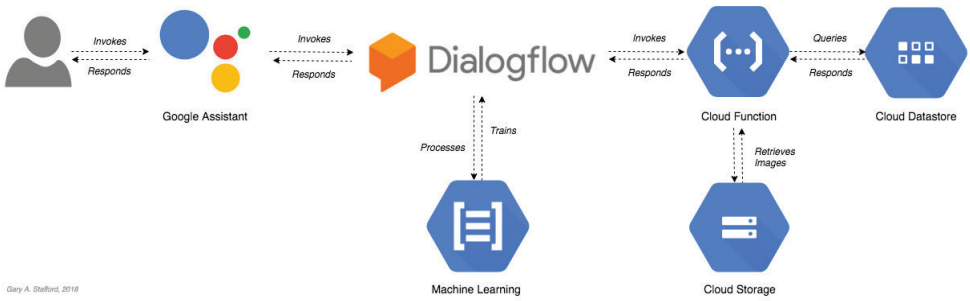


Figura 4: Arquitetura de uma aplicação para o Google Assistant[15].

É ainda interessante o facto da Google não fechar as possibilidades aos programadores permitindo sempre alternativas ao uso das suas ferramentas, oferecendo toda a flexibilidade necessária. É possível utilizar o Actions SDK para os que, por exemplo, não pretenderem utilizar o Dialogflow, ficando assim encarregues pela construção do modelo de conversação em formato JSON. Por outro lado, o próprio Google Assistant não é limitado ao uso por parte de produtos da Google, é possível utilizar o mesmo em qualquer dispositivo Android e/ou iOS, e ainda outros dispositivos. Exemplo disto é o facto da própria documentação explicar como seria possível ter o assistente digital embebido num Raspberry Pi.

Disponibiliza ainda uma documentação muito completa, sucinta e com diversos exemplos e alternativas, permitindo que qualquer programador se sinta confortável ao explorar as diferentes funcionalidades oferecidas para a construção de uma aplicação para o assistente.

2.2.4 *Cortana by Microsoft*

O caso da Microsoft é um pouco distinto dos restantes, já que para além de disponibilizar um kit para construção de aplicações para o seu assistente digital, a Cortana, disponibiliza ainda uma framework bastante complexa para construção de bots para qualquer plataforma - *Bot framework*[7]. Esta framework oferece SDKs para .NET e Node.JS, integrações com o LUIS[8] para Language Understanding, com o QnA Maker[9], para construção fácil e rápida de Q&A (perguntas e respostas) e ainda com os seus serviços na cloud, onde estas aplicações podem ficar alojadas.

7 https://dev.botframework.com/
8 https://www.luis.ai
9 https://www.qnamaker.ai

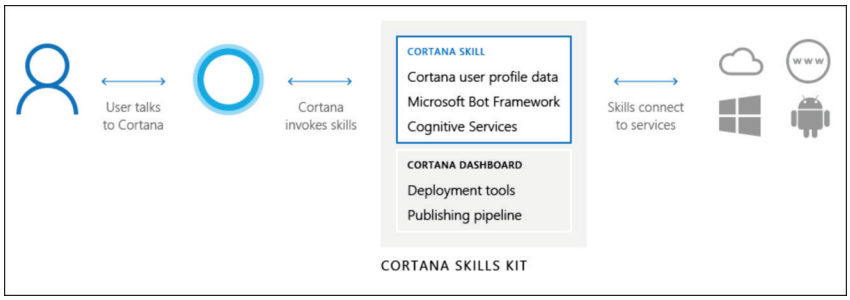


Figura 5: Diagrama explicativo do funcionamento de aplicações para a Cortana[23].

Apesar do SDK da Microsoft ter sido dos primeiros a estar disponível para utilização[6], do ponto de vista do pacote oferecido para desenvolvimento de aplicações para a Cortana, enfrentamos os mesmos desafios que no ecossistema Apple, ficando bastante limitados pelo desenvolvimento de aplicações com base em ferramentas disponibilizadas pela Microsoft, relembrando ainda que o seu assistente digital funcionaria apenas em produtos com sistemas operativos Windows.

2.2.5 *Bixby by Samsung*

O impacto que a Samsung tem atualmente no mercado tecnológico é claramente notável, daí apresentar também o seu próprio assistente digital embutido em qualquer dos seus smartphones/tablets e até, já anunciado, nos seus frigoríficos inteligentes - Family Hub[10]. Apesar de ser um dos assistentes digitais mais utilizados, capaz de facilmente fazer concorrência à Siri ou ao Google Assistant, o Bixby ainda não disponibiliza a possibilidade de criação de aplicações por parte de terceiros.

Devemos, no entanto, estar conscientes que de todos apresentados a Samsung é a empresa que disponibiliza uma maior gama de dispositivos eletrónicos (smartphones, tablets, tvs, frigoríficos, etc) e ainda que o seu foco está direcionado para o desenvolvimento de uma *connected home*, onde certamente o seu assistente digital terá grande impacto.

2.3 Âmbito da aplicação

Para o contexto do presente tema da dissertação apenas será possível explorar um dos assistentes digitais anteriormente apresentados, no entanto encontramos facilmente características semelhantes entre as diferentes soluções oferecidas. O esqueleto em que as aplicações para os assistentes assenta é muito semelhante sendo que as diferenças que se

10 https://www.samsung.com/us/explore/family-hub-refrigerator/overview/

vão destacando estão normalmente associadas aos tipos de kits para desenvolvimento que cada uma das soluções oferece.

Apesar de grande oferta de soluções, é possível reconhecer que a Amazon e a Google oferecem um conjunto de funcionalidades mais completas e complexas que os concorrentes, oferecendo ferramentas para solucionar as diversas barreiras inerentes à construção de aplicações para um assistente digital e ainda alternativas a estas ferramentas, dando aos programadores toda a flexibilidade que possam vir a precisar[27].

Entre os dois que se destacam, a decisão final foi o Google Assistant. Esta decisão foi tomada de acordo com dois pontos considerados essenciais para o futuro desenvolvimento de aplicações para o assistente:

- oferta de um ecossistema completo que permite desde a construção do modelo de conversação até ao próprio serviço para dar host da aplicação na cloud - Dialogflow, Actions, Firebase, etc.
- documentação rigorosamente detalhada, oferecendo alternativas para muitos casos particulares, em conjunto com uma maior e mais desenvolvida comunidade.
- satisfação dos consumidores quando comparado com as restantes alternativas[30].

2.4 ESTADO DA ARTE

Ao longo deste capítulo já foi apresentado o estado atual ao nível das tecnologias, plataformas e ecossistemas que permitem o desenvolvimento de assistentes digitais. Nesta secção serão apresentados artigos resultantes de investigação tanto de arquiteturas para os assistentes, como ao nível da gestão de dispositivos IoT.

Em *Singular Adaptive Multi-Role Intelligent Personal Assistant for Human Computer Interaction*[20] é evidenciada a importância que os assistentes digitais têm e ainda mais a que possivelmente terão no futuro, dizendo até que apesar de não se saber como irá a inteligência artificial transformar o mundo, existe um consenso de que será através destes assistentes onde terá maior visibilidade, fazendo com que a utilização de teclados ou outro tipo de dispositivos de interação com aplicações se torne obsoleta.

O artigo propõe uma arquitetura para assistentes que para além das tarefas básicas habituais destes (e.g. consultar o tempo ou pesquisar informação) seja também capaz de recolher informação através de dispositivos IoT, fazendo o seu processamento e mudando o comportamento perante os utilizadores conforme a informação recolhida. São apresentados alguns dos maiores desafios, ao nível de engenharia, que têm de ser ultrapassados para que seja possível proliferar o uso dos assistentes, dentro dos quais estão a falta de

protocolos de comunicação entre dispositivos IoT padrão, a segurança e a privacidade dos utilizadores.

Na arquitetura proposta no artigo podemos contar com 3 módulos distintos:

- *HCI Unit*: *HCI* é *Human-Computer interaction*, o objetivo desta unidade é ser próxima do utilizador, na medida em que será o ponto de comunicação com este. Nesta unidade poderemos contar com componentes de *Speech To Text*, isto é, tradução de voz em texto; assim como *Natural Language Understanding*, isto é entender o que o utilizador quer efetivamente dizer, através reconhecimento de intenções e entidades.

- *Integration Unit*: unidade de integração do assistente com aplicações e dispositivos IoT. Os protocolos de comunicação com os dispositivos devem ser flexíveis pelo que, é esperado que esta unidade contenha múltiplos componentes para comunicação através de APIs REST e/ou protocolos de messagens (segundo uma arquitetura *publisher-subscribe*).

- *Adaptive Intelligence Unit*: por fim, a unidade central, responsável por aplicar ingeligência artifical e *Machine Learning*, para aprender as diferentes formas de interação do utilizador e garantir que, em conjunto com as duas outras unidades, o sistema funcione corretamente como um todo.

Por fim, é ainda apresentada uma prova de conceito desta arquitetura na prática, embora muito direcionada para a primeira unidade da arquitetura anterior - *HCI Unit* - cujo foco está na criação de uma experiência de conversação com os utilizadores. Assim sendo esta prova de conceito tira partido da plataforma *Dialogflow* (que também será utilizada na presente dissertação) para construir uma aplicação *BikeShop-Agent* que permite aos seus utilizadores não só saber o horário de funcionamento como também agendar uma marcação para fazer uma reparação. Nesta prova de conceito são exploradas diversas funcionalidades da plataforma Dialogflow, dentro das quais a definição de intenções, extração de entidades e controlo de fluxo de diálogos que também serão exploradas nesta dissertação.

Enquanto que o artigo anterior tem o foco para os assistentes digitais, o artigo *IoT device management framework for smart home scenarios*[26] é mais focado na outra vertente dissertação, a gestão de dispositivos IoT no mesmo ambiente. O artigo propõe a arquitetura de uma framework que permita a gestão destes dispositivos dentro do contexto *smart home*.

Este artigo começa por referir a importância da utilização dos dispositivos IoT para o contexto *smart home* e como se espera que a utilização destes dispositivos venha a ser cada vez mais comum no futuro. São ainda destacadas as dificuldades que surgem em sistemas que utilizam vários destes tipos de dispositivos devido à heterogeneidade tanto dos dispositivos como dos dados que são utilizados na comunicação entre estes.

A framework apresentada é constituída por 3 camadas distintas:

- *Proxy layer*: camada que serve de *proxy* e permite a comunicação com os dispositivos através de diferentes tipos de protocolos e tecnologias (bluetooth, NFC, Wi-Fi etc).
- *Device Management layer*: camada para gestão dos dispositivos, é composta por diferentes serviços, sendo que entre estes estão presentes serviços para configuração dos dispositivos; apresentação de informação sobre os dispositivos presentes no mesmo ambiente (e.g. identificador do dispositivo, *endpoint* para comunicação com o mesmo, etc); e ainda recolha automática dos seus dados através de sensores.
- *Service Enablement Layer*: as funcionalidades da camada de gestão de dispositivos deverão ser expostas através de uma API RESTful, através da utilização de serviços web. É esta camada que garante a interoperabilidade de outros dispositivos (e.g. *smartphones*) com os dispositivos IoT. Questão de políticas de autorização sobre os dispositivos que os utilizadores podem ou não controlar devem também ser incluídas nesta camada.

Pelos artigos [21], [31] e [25] é possível verificar que a utilização de um Raspberry Pi ou dispositivo similar é muito comum para investigações dentro do domínio dos dispositivos IoT e automatização destes dentro do contexto *smart home*. Desta forma, no desenvolvimento do sistema resultante será também utilizado um Raspberry Pi para, através dos seus periféricos, simular o comportamento dos dispositivos que serão controlados.

3

DESENVOLVIMENTO DE APLICAÇÕES PARA UM ASSISTENTE DIGITAL

No presente capítulo será detalhado o processo de especificação e desenvolvimento de aplicações para um assistente digital. Notemos que não serão detalhadas soluções parametrizadas para um assistente específico, já que a ideia passará mesmo por abordar, de uma perspetiva geral, a construção de uma aplicação/extensão para um dos quaisquer assistentes digitais existentes.

De modo a facilitar a compreensão de cada um dos passos que constituem o desenvolvimento de aplicações para um assistente digital acompanharemos cada um dos passos com um exemplo de uma aplicação trivial. Consideremos então uma aplicação *GeekNum* cujo objetivo é dar factos interessantes sobre números aos seus utilizadores.

3.1 DESIGN DA APLICAÇÃO

Como sabemos a interface das aplicações que pretendemos construir passa então a ser completamente distinta das UI a que estamos habituados, tanto para aplicações web, mobile ou desktop. Nestas aplicações a interação com os utilizadores é predominantemente através da voz/som, embora seja possível fazer uso dos ecrãs dos mais diversos dispositivos onde estes assistentes se inserem para representar a informação necessária, segundo diferentes formatos de media. Assim sendo, torna-se necessário criar uma experiência, através de conversação, que entusiasme o utilizador final.

3.1.1 *Seleção de casos de uso*

Inicialmente é preciso definir quais os casos de uso que a aplicação terá, isto é, as diferentes funcionalidades que oferecerá ao utilizador. É de notar que um caso de uso deve ser uma operação simples, intuitiva e capaz de conduzir a uma resposta relativamente rápida.

Este processo deve incluir uma fase posterior para recolha de requisitos e ainda identificação dos tipos de utilizadores, tal como uma aplicação normal. Com esta informação conseguimos selecionar os casos de uso que consideramos intrínsecos aos objetivos funcionais da aplicação. Notemos que, cada aplicação tem funcionalidades e objetivos característicos, não existe a necessidade de tentar agrupar todo o tipo de casos de uso numa só aplicação quando tal pode ser dividido por diversas aplicações e estas inseridas no mesmo assistente digital. Deste modo, esta seleção deve ser cautelosa e limitada, já que será daqui que todo o processo de desenvolvimento da aplicação partirá.

Para o exemplo da aplicação *GeekNum* a seleção de casos de uso seria extremamente simples já que estamos a considerar uma aplicação cuja única funcionalidade seria dar factos interessantes sobre um dado número fornecido pelo utilizador. No entanto, esta deveria ser devidamente moldada de acordo com o seu público-alvo, já que a abordagem da aplicação deverá ser completamente distinta ao tratar-se de uma aplicação para crianças, por exemplo.

3.1.2 *Criação de uma persona*

O sucesso de aplicações para os assistentes é caracterizado pela proximidade que estas têm com comportamentos do ser-humano[7], tornando a criação de uma persona um fator muito importante. Uma persona é uma representação fictícia de uma pessoa. Na criação de uma persona devem-se ter em conta informações e características específicas da mesma para que de acordo com estas seja possível construir a sua personalidade.

A aplicação pretende simular uma conversa, algo que acontece entre duas ou mais pessoas, mas que agora passará a existir entre uma pessoa (utilizador) e um dispositivo (assistente digital). Cada pessoa tem uma personalidade distinta e esta manipula, intrinsecamente, a maneira de falar de cada ser humano. A criação de uma personalidade para a aplicação torna-se assim um aspeto muito importante já que queremos que, ao longo do fluxo das diversas conversas com os utilizadores, esta mantenha uma forma consistente de falar e comportar-se.

Para criar uma persona com sucesso é necessário reconhecer o tipo de utilizador que estará a comunicar com a mesma e ainda perceber a situação em que este se encontra, perceber qual é a necessidade do mesmo ao utilizar a aplicação - os seus objetivos e necessidades. Assim que entendemos estes pontos podemos criar uma persona que seja de facto relevante durante toda a conversa com o utilizador, já que consegue oferecer respostas aos diversos pedidos por ele realizados.

Tendo em conta os pontos anteriores, um exemplo de persona que poderia retratar a aplicação *GeekNum* é um robot caracterizado pela intelectualidade com que aborda os seus

utilizadores, relembrando que, afinal de contas, este possuí conhecimento sobre inúmeros factos para qualquer número que lhe é apresentado. Transparecer uma entidade culta e inteligente deverá ser uma das prioridades na persona que a aplicação revela para os seus utilizadores.

3.1.3 *Escrita de diálogos*

O fluxo de uma conversa é extremamente complexo de traduzir para o mundo digital. Nenhuma resposta dada pelo o utilizador pode ser prevista, mesmo quando a pergunta que a gera é demasiado concreta. Enquanto que uma UI tradicional limita as respostas ao utilizador, ao utilizar a voz como interface isto deixa de ser possível - um exemplo seria pedir ao utilizador a sua data de nascimento, enquanto numa UI comum existem três parâmetros para preencher (dia, mês e ano), ao utilizar a voz passam a ser possíveis inúmeros tipos de resposta possíveis.

Além de ter de lidar com toda esta multiplicidade e complexidade de respostas por parte dos utilizadores, é necessário tentar abranger os diversos fluxos que uma conversa pode ter, em particular, os casos em que o utilizador pode estar a fugir completamente ao fluxo ideal para concretizar um determinado caso de uso.

Na Figura 6 apresenta-se um exemplo de um diálogo entre um utilizador e a aplicação *GeekNum*. Na figura vemos que a aplicação está pronta para lidar com dois tipos de resposta por parte do utilizador (caso ele pretenda um facto adicional ou não). Verificamos ainda que se a aplicação não reconhece a resposta do utilizador, ou seja um desvio ao fluxo esperado, avisa-o e repete a pergunta inicial.

Regra geral, os diálogos do utilizador deverão começar por indicar qual a aplicação com a qual querem interagir, enquanto que o primeiro contacto por parte das aplicações será uma breve apresentação onde se enumeram as funcionalidades relevantes que oferecem. A partir desta apresentação o utilizador faz um ou vários pedidos à aplicação, sendo que esta vai tentando sequencialmente recolher toda a informação necessária para conseguir satisfazer o seu pedido. Dado este por realizado, as aplicações costumam perguntar ao utilizador se pretende utilizar outro tipo de funcionalidade, não deixando que a conversa entre ambos acabe abruptamente.

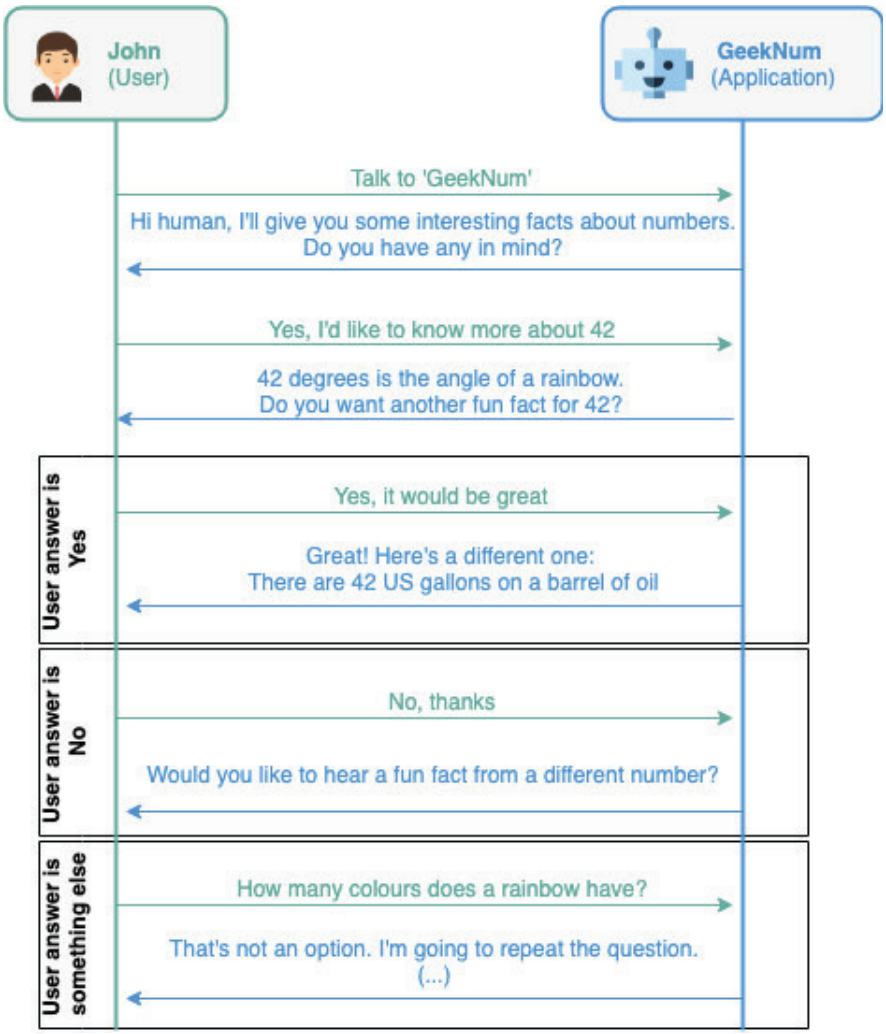


Figura 6: Exemplo do diálogo de um utilizador com uma aplicação.

Para escrever exemplos de diálogos possíveis devemos focar-nos nos cenários de sucesso, também conhecidos como *happy paths*, estes assumem que não existem problemas técnicos e que os utilizadores seguem o fluxo expectável de conversação. Dados estes por escritos podemos partir para a escrita das situações em que o utilizador segue uma direção inesperada e/ou precisa de ajuda do assistente.

Saudação

Inicialmente devemos escrever um conjunto de frases de saudação, assim que os utilizadores invocam a aplicação. Esta é a primeira oportunidade para estabelecer uma relação com o utilizador, demonstrando algumas das características que foram anteriormente definidas para a persona da aplicação. Aqui devemos saudar o utilizador apresentando, muito sucin-

tamente, o(s) principal(ais) objetivo(s) da aplicação. A primeira interação do utilizador com a aplicação *GeekNum* na Figura 6 é um exemplo de uma frase de saudação.

Cenários de sucesso

De seguida devemos construir diálogos para o que consideramos ser o fluxo ideal que o utilizador deve seguir para conseguir realizar, com sucesso, um dos casos de uso oferecidos. Para as respostas devemos oferecer um conjunto de alternativas, de modo a que a utilização contínua da aplicação não se torne aborrecida. Fazendo analogia ao ser humano, apesar de termos conversas com temas idênticos não estamos à espera de ouvir, por parte de outras pessoas, sempre a mesma resposta literal. Na Figura 6 tanto quando o utilizador responde *sim* como quando responde *não*, ambas as respostas estão incluídas dentro dos *happy paths*, já que é algo que a aplicação pode esperar como resposta.

Estratégias para recuperar o fluxo

O fluxo da conversa nem sempre corre como esperado: o utilizador pode não ter percebido o que era requisitado; pode ter deixado de responder ou o assistente pode não perceber a resposta do utilizador. Por isso devemos sempre oferecer estratégias para voltar ao fluxo, estas estratégias geralmente envolvem repetir a pergunta inicial oferecendo mais detalhes ou explicar ao utilizador o ponto em que se encontra atualmente na conversa. Quando o utilizador pergunta à aplicação *GeekNum* quantas cores tem o arco-íris, não está a responder à pergunta inicialmente feita pela aplicação, logo está a desviar-se do fluxo inicialmente idealizado.

3.1.4 *Construção iterativa*

A fase seguinte será testar os exemplos de diálogos construídos com utilizadores reais, através da utilização da voz. Reconhecer se de facto o fluxo e as próprias frases se adequam às interações que o ser humano está acostumado a ter.

Por fim podemos começar a desenvolver a aplicação, com base em tudo o que foi especificado anteriormente. É de notar que este processo deve ser iterativo, o programador deve procurar constantemente melhores soluções para o diálogo, de modo a que este se adeque aos *use-cases* que a aplicação oferece ao utilizador ou simplesmente atualizar devidamente o conjunto de respostas oferecidas num determinado ponto do fluxo, uma vez que se o assistente der sempre as mesmas respostas o utilizador ficará aborrecido e o mais certo será deixar de usar a aplicação.

3.2 DESENVOLVIMENTO

Dada por concluída a especificação da aplicação é possível então começar a pensar em todo o processo de desenvolvimento da mesma. Abordaremos, de seguida, todos os processos e componentes necessários ao correto funcionamento de uma aplicação deste domínio.

3.2.1 *Natural Language Processing*

O tipo de *user-interface* que caracteriza estas aplicações apresenta múltiplos componentes imprescindíveis ao correto funcionamento das mesmas, sendo a tradução de voz em texto e vice-versa exemplos destes. Para um assistente digital entender os pedidos dos utilizadores é necessário que numa fase inicial exista uma conversão da voz do utilizador para texto, o qual será processado e mais tarde será originada uma resposta, em formato texto, que agora terá de ser traduzida para voz, já que os assistentes digitais modernos falam, literalmente, com os seus utilizadores.

Speech-to-text

O componente de *Speech-to-text*[1] ou *Speech recognition* é extremamente importante já que, enquanto nas UIs comuns os utilizadores são obrigados a escrever texto que só poderá depender da língua do utilizador, com a utilização de interação por voz existem muitos outros fatores a ter em conta para além da língua. Todos os seres humanos falam de forma diferente, tanto devido a sotaques como diferentes tons de voz, esta acaba por ser uma característica intrínseca a cada ser humano, o que torna todo este processo de tradução em texto mais complicado.

Este componente permite aos assistentes entender utilizadores provenientes de qualquer parte do mundo, com línguas e tons de voz distintos, fazendo a tradução do que dizem em texto. Este texto, ao contrário do som das falas dos utilizadores, já poderá ser processado diretamente.

Natural Language Understanding

A transformação de voz em texto é só o primeiro passo, agora será necessário que o assistente entenda o significado do texto, em particular, a intenção do utilizador com aquela frase ou conjunto de frases. *Natural Language Understanding* é o componente com esta responsabilidade. Regra geral os NLU permitem, através da utilização de Inteligência Artificial reduzir as sentenças dos utilizadores a um conjunto de intenções e entidades, sendo isto

1 https://www.techopedia.com/definition/23767/speech-to-text-software

possível através do fornecimento de exemplos de conversação que permitem, com ajuda de *Machine Learning*, aprender os possíveis diálogos com os utilizadores[2].

Na Figura 7 podemos observar o principal papel do NLU. Através de uma mensagem do utilizador consegue extrair as suas intenções e entidades, que são aquilo que é realmente necessário para perceber os seus objetivos.

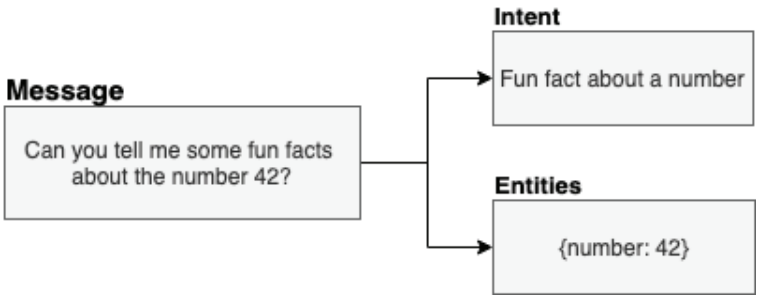


Figura 7: Exemplo da tradução de uma mensagem em intenções e entidades.

Nas intenções devemos ter em conta o que os utilizadores pretendem fazer e se a nossa aplicação será capaz de satisfazer estas necessidades. As intenções identificadas irão determinar o fluxo do diálogo que o utilizador terá e ainda as aplicações que serão invocadas para satisfazer o seu pedido. Por outro lado, as entidades representam um termo ou um objeto no input do utilizador que providencia mais informação ou um contexto específico para uma intenção particular. As entidades tornam possível uma única intenção representar múltiplas ações, já que são estas que completam o pedido do utilizador. Regra geral, as intenções estão associadas a verbos (algo que o utilizador quer fazer), enquanto as entidades a nomes.

Toda a interação com os utilizadores acaba por ser um diálogo entre estes e o assistente, será ainda necessário manter um contexto para que o modelo de conversação faça sentido - se o utilizador faz uma pergunta o assistente deve conseguir responder a essa mesma pergunta e ter noção do contexto de conversação atual para dar a sua resposta. Os seres humanos são imprevisíveis, logo será necessário criar um modelo de conversação que tenha diferentes ramificações, permitindo sempre alternativas de diálogo entre o assistente e o utilizador. Quando o utilizador faz um pedido, e já depois deste ser completamente entendido pelo assistente deverá ser realizada uma invocação à aplicação responsável por satisfazer este pedido. Esta invocação dará origem a uma resposta, em formato de texto, que terá de ser comunicada ao utilizador.

2 https://searchenterpriseai.techtarget.com/definition/natural-language-understanding-NLU

Text-to-Speech

Sabemos bem que a interação dos utilizadores com o assistente por voz é idealmente bidirecional, apesar da opção por texto estar presente em praticamente todos os assistentes digitais que foram anteriormente referidos. Assim sendo, será necessário o assistente conseguir transformar a resposta de texto, conseguida pela invocação à aplicação, para voz. É aqui que entra o componente de *Text-to-speech*, responsável por transformar as respostas em linguagem entendida por computadores em voz artificial para dar uma resposta mais realista ao utilizador. Atualmente já é possível simular emoções no tom de voz do assistente através da utilização de, por exemplo, *Speech Synthesis Markup Language* (SSML).

Como podemos ver o componente NLP acaba por ser muito complexo e é uma das principais razões pelas quais os assistentes digitais ainda se encontram numa fase muito preliminar. Existem, porém, ferramentas bastante completas que permitem simplificar todo este processo anteriormente relatado, tais como IBM Watson ou Dialogflow.

3.2.2 *Aplicações/Serviços*

Depois de reconhecidas as intenções e entidades do utilizador para um determinado pedido passa a ser necessário efetuar a ação respetiva utilizando-as como parâmetros decisivos, aqui entra a lógica de negócio e a aplicação propriamente dita. O desenvolvimento da lógica de negócio deve seguir as metodologias e processos que vão de encontro com o definido nas documentações destes assistentes digitais.

A comunicação com estas aplicações é regra geral através de pedidos HTTP, em particular pedidos POST cujo formato e conteúdo possa variar conforme o assistente onde a aplicação se irá inserir. O conteúdo dos pedidos que chegam à aplicação traduz toda a informação que o componente anterior, o NLP, conseguiu recolher e, de certa forma, processar. Cabe agora à aplicação interpretar esta informação de modo a que o pedido do utilizador seja satisfeito. Esta camada contém diretrizes similares às das aplicações web tradicionais com algumas adições que variam desde o controlo do contexto de conversação às formas de apresentação da informação segundo formatos media distintos (voz, vídeo, imagem ou texto). Sistemas de autenticação e acessos a bases de dados são outros exemplos de componentes que se encaixam nesta camada cujo objetivo é englobar todo o *back-end* da aplicação para o assistente.

Grande parte dos serviços que se desenvolvem para responder a pedidos provenientes dos utilizadores recorrem a informação de outros serviços externos para conseguir respostas apropriadas e adequadas ao pedido inicial. Um exemplo muito simples seria a aplicação

GeekNum necessitar de efetuar um pedido a um serviço externo para saber factos interessantes para um determinado número, já que esta informação não tem, necessariamente, de ser da responsabilidade da aplicação, pode simplesmente ser fornecida através de um serviço externo que expõem uma API para tal. Desta forma, os serviços externos acabam por ser uma componente enriquecedora para aplicações deste domínio.

Regra geral, a arquitetura desta camada da aplicação é dividida em três módulos com responsabilidades distintas.

Natural Language Processing

Este módulo tem contacto direto com o componente NLP e deve, por essa mesma razão, adquirir o contexto de conversação em conjunto com as diversas intenções e entidades recolhidas pelo NLP para redirecionar o fluxo da aplicação para a componente indicada. A partir do momento em que a aplicação reconhece a intenção do utilizador pode invocar a componente da lógica de negócio que a satisfaça.

Lógica de negócio

Depois de invocada, a componente de lógica de negócio é responsável por efetuar o processamento necessário à resolução de uma solução para o pedido inicial do utilizador. Este processamento exige, grande parte das vezes, a invocação de serviços externos onde é adquirida informação adicional para o satisfazer.

Construção da resposta

Assim que é conseguida toda a informação é necessário construir uma resposta para o utilizador tirando proveito da mesma. A construção de uma resposta pode incluir a apresentação de diversos tipos de informação em formato visual (se o assistente disponibilizar um ecrã) ou simplesmente através de som. Muitas plataformas oferecem já controlo para a voz do assistente, oferendo funcionalidades para, de certo modo, seja possível controlar as suas emoções através da voz, permitindo que o assistente consiga responder ao utilizador com diferentes níveis de intensidade.

3.2.3 *Testes*

Sendo os assistentes digitais uma tecnologia relativamente recente, as alternativas para os testar são, deste modo, muito limitadas. Geralmente as plataformas onde estes se inserem oferecem ferramentas para testar o seu próprio assistente digital, a Amazon oferece o

Echosim[3] enquanto a Google o Actions Simulator[4]. O processo de desenvolvimento deve ser iterativo e intercalado com testes, de modo que seja possível lutar por uma interação assistente-utilizador realista conseguindo-se mais facilmente atrair o público-alvo.

3.3 DEPLOY

Os processos anteriormente descritos devem ter em conta que as aplicações para os assistentes terão mais de tarde de ser submetidas para a *cloud*, onde ficam alojadas e prontas a utilizar pelos seus utilizadores.

As aplicações serão invocadas através de um dispositivo físico (Google Home, Amazon Echo, etc) e não seria prático que cada aplicação tivesse de ser manualmente instalada no mesmo. Assim as aplicações devem estar disponíveis para receber invocações, através de pedidos HTTP, por estes dispositivos. Regra geral estas aplicações são implementadas através de webhooks - lógica de negócio que é executada quando um determinado callback é verificado, assumindo que durante esta execução é realizado um pedido HTTP. Assim, estes webhooks devem ficar alojados na *cloud* e ser acessíveis a qualquer momento por invocações que, por norma, partirão do dispositivo físico.

3 https://echosim.io
4 https://developers.google.com/actions/tools/simulator

4

DESENVOLVIMENTO DE APLICAÇÕES PARA O GOOGLE ASSISTANT

Google Assistant, o assistente pessoal virtual da Google é já utilizado em mais de 500 milhões de dispositivos, incluido smart speakers, smartphones, carros, TVs, headphones, smartwatches, etc. O desenvolvimento de aplicações para extensão de funcionalidades deste assistente é realizado através da plataforma *Actions on Google* - uma plataforma direcionada para programadores cujo objetivo seja criar e gerir experiências de conversação aliciantes para os utilizadores do assistente.

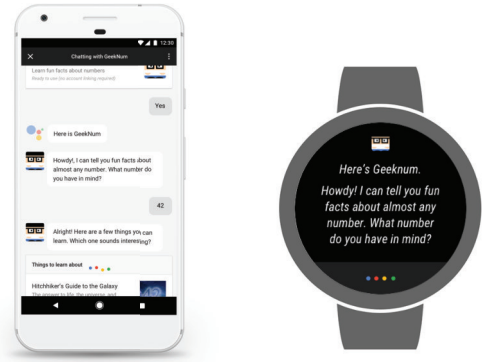


Figura 8: Exemplo da utilização do Google Assistant em múltiplos dispositivos. [1]

Ao contrário das tradicionais aplicações mobile e desktop, aqui os utilizadores interagem com *Actions* para o assistente, através de som/voz, simulando uma conversa entre assistente e utilizador.

1 Imagem retirada da documentação do *Actions on Google*.

4.1 TERMOS-CHAVE

Seguem-se alguns termos-chave, para auxiliar a compreensão de como é realizado o desenvolvimento de uma aplicação para o assistente[2].

- **Action**: *action* ou ação, é o ponto de entrada para uma interação que foi construída para o assistente, suporta uma **intent** e tem o correspondente processamento necessário para a satisfazer (**fulfilment**).
- **Intent**: *intent* ou intenção é um objetivo ou tarefa subentendida que os utilizadores pretendam realizar, tal como encomendar uma refeição ou encontrar uma música específica. Na plataforma *Actions on Google*, cada intenção é representada com um identificador único e os correspondentes enunciados/falas dos utilizadores que a podem despertar.
- **Fulfillment**: *fulfillment* é um serviço, aplicação ou qualquer outro tipo de lógica que lida com uma intenção e realiza a ação correspondente.
- **Actions project**: um projeto criado na plataforma *Actions on Google* que deverá conter pelo menos uma ação, para a *welcome intent*, mas pode conter ações adicionais para responder a outras intenções relevantes. Este projeto é criado para gerir, testar e publicar uma coleção de ações. A Google mantém estes *actions projects* na cloud.
- **Actions Console**: ferramenta que auxilia o desenvolvimento de ações, já que permite a criação, manutenção, teste e publicação das mesmas. Esta ferramenta pertence à plataforma *Actions on Google*.

4.2 TIPOS DE AÇÕES

As ações são a base para possibilitar a extensão de funcionalidades ao assistente. Através da plataforma *Actions on Google* é possível a construção de ações de dois tipos distintos:

Conversational Actions

Ações de conversação ou *Conversational Actions* são ações cuja funcionalidade é criar algum tipo de conversação entre utilizadores e o assistente. Para iniciar este tipo de ação, o utilizador deve invocar a mesma através do assistente, normalmente através de uma frase - *Hey Google, talk to GeekNum* - permitindo facilmente o assistente reconhecer o nome da ação com a qual o utilizador pretende interagir, neste caso a ação *GeekNum*. A partir deste

2 Tendo em conta a semântica dos termos alguns destes serão utilizados em inglês ao longo do presente documento

ponto, o utilizador fala para a ação, dando-lhe input ao qual desta deve ser capaz de responder, através de um diálogo bidirecional, até que o pedido do utilizador seja satisfeito ou a conversa simplesmente tenha terminado.

Smart home Actions

Ações de smart home ou *Smart home actions* são ações que permitem controlar dispositivos de Internet of Things (IoT) através do assistente. A construção destas ações permite os utilizadores conectarem-se, efetuarem queries e controlarem dispositivos através de uma infraestrutura existente na *cloud*.

Apesar do objetivo final da dissertação ir ao encontro do segundo tipo de ações - *smart home actions* - estas são simplesmente uma abstração das *conversational actions* para um propósito mais específico. Assim sendo, faz sentido explorar o processo de desenvolvimento de ações dos dois tipos.

4.3 FLUXO DOS PEDIDOS

Para melhor compreensão do funcionamento do assistente, em conjunto com a interação deste com as diversas plataformas oferecidas no ecossistema, explicaremos, com detalhe, o fluxo dos pedidos dos utilizadores.

Numa fase inicial o utilizador terá de fazer um primeiro pedido onde especifica a ação com a qual pretende estabelecer contacto. Cada ação tem um propósito diferente, como se fosse uma aplicação distinta - na Figura 9, retirada da documentação do *Actions on Google*[15], verificamos que o utilizador irá interagir com a ação com o nome *Which Language*. As colunas representadas na figura correspondem, da esquerda para a direita: ao utilizador, às respostas do Google Assistant (que podem ser através do dispositivo Google Home), ao próprio Google Assistant, à plataforma Dialogflow e por fim ao *fulfillment* da ação. Assim que o Google Assistant recebe este pedido, irá traduzir o pedido de voz para texto, já que só assim será possível fazer o seu processamento para recolha de entidades e intenções, através da utilização de *Machine Learning* aplicada às frases exemplo dos utilizadores fornecidas. Recolhida esta informação, passa a ser possível fazer a identificação da ação correspondente e desta forma despertar a intenção a esta associada no Dialogflow - plataforma responsável pela identificação de intenções e entidades no pedido do utilizador (NLU) que será explorada com mais detalhe na secção 4.6. O pedido do utilizador não tem informação adicional para além do nome da ação que quer invocar, como tal a intenção a despertar será, por defeito, a intenção de *Welcome*, isto é, de boas-vindas.

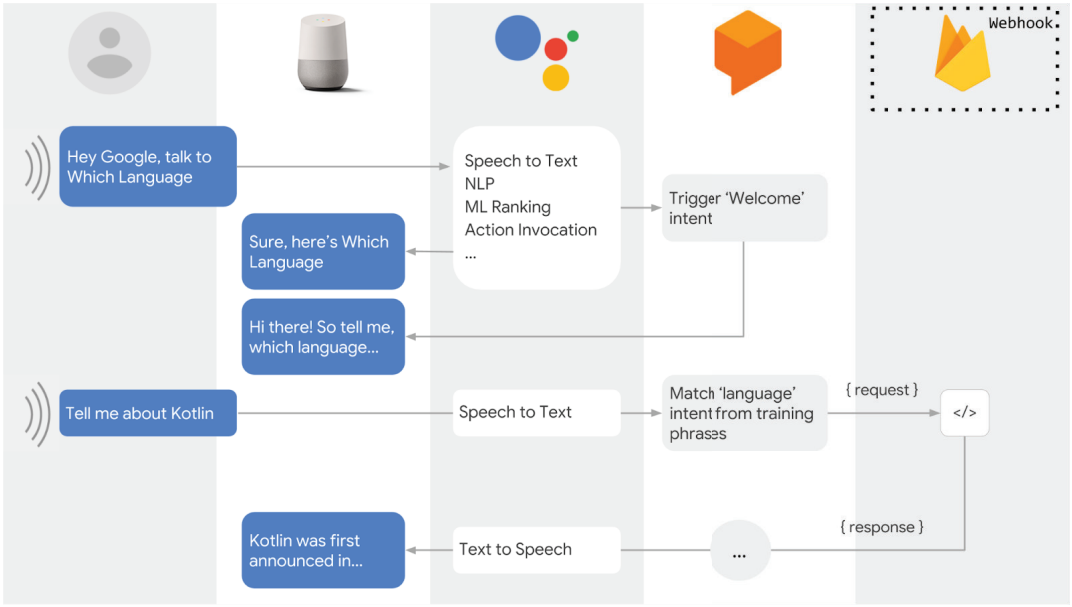


Figura 9: Arquitetura e fluxo dos pedidos. [3]

A partir deste momento o utilizador estará a comunicar com esta ação em específico, mas não é por isso que os pedidos deixam de ser processados pelo Google Assistant, relembremos que é nesta etapa que, por exemplo, é realizada a conversão do pedido de voz em texto. No caso particular do pedido *Tell me about Kotlin* podemos verificar, na coluna do Dialogflow, que foi identificada uma correspondência com a intenção *language*, algo que só foi possível já que foram previamente fornecidas frases para treinar o assistente. Notemos ainda que esta intenção, tal como grande partes das intenções desenvolvidas para o assistente, necessitou de fazer um pedido a um serviço externo - *fulfillment* - e a resposta deste foi, no Google Assistant, convertida de texto para voz e enviada ao utilizador através do dispositivo Google Home.

4.4 Interação dos utilizadores com ações

Para auxiliar a compreensão de como os utilizadores interagem com uma ação, usaremos, uma vez mais, a ação *GeekNum*. Os utilizadores podem invocar esta ação através do assistente no seu smartphone ou smartwatch para aprender factos interessantes sobre qualquer número. Na Figura 10 é demonstrado um exemplo da utilização da ação *GeekNum*.

3 Imagem retirada da documentação do *Actions on Google*.

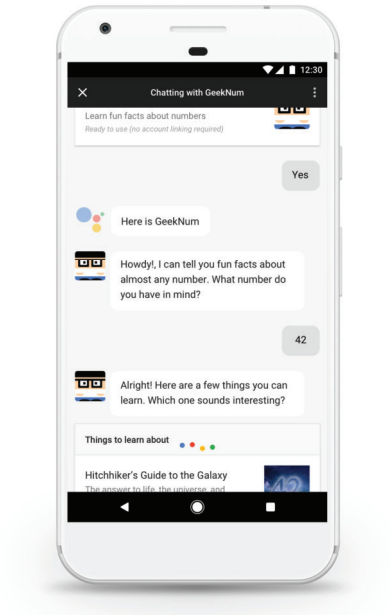


Figura 10: Exemplo da utilização da ação GeekNum.

1. Um utilizador interage com o assistente fazendo o pedido: *Hi Google, I want to learn about numbers*. Com base neste pedido o assistente procura pela ação que melhor poderá satisfazer a intenção do utilizador e retorna uma ação com o nome *GeekNum*.

2. O assistente pergunta ao utilizador se esta é, de facto, a ação que pretende invocar, ao qual o utilizador responde afirmativamente - *Yes*. O assistente envia para o *fulfillment* desta ação um pedido HTTPS POST e recebe uma resposta, a qual apresenta ao utilizador.

3. O assistente apresenta esta resposta na user interface que o utilizador está a utilizar, que tanto pode ser um ecrã de um smartphone/smartwatch, como simplesmente através de som, para dispositivos que não têm um ecrã, como é o caso do próprio Google Home. Agora o assistente introduz o *GeekNum*, a mensagem de apresentação é precisamente a resposta que o assistente recebeu com o pedido ao *fulfillment*. A partir deste momento o assistente entrega o utilizador ao *GeekNum*, podemos admitir que começa aqui a conversa entre estes.

4. Durante esta conversa, o assistente envia a mensagem do utilizador para o *fulfillment* que será responsável por responder diretamente para o assistente, isto é representado na Figura 11.

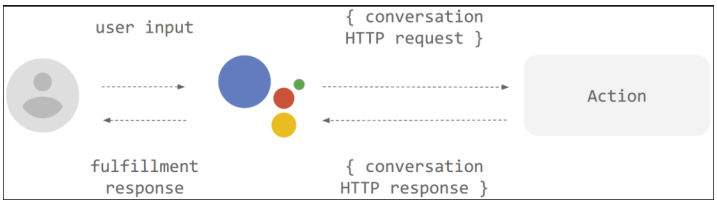


Figura 11: Exemplo da invocação da ação e respectiva resposta da mesma.

5. A conversa continua até ao ponto em que o *fulfillment* tem recolhida toda a informação necessária para completar o pedido inicial do utilizador.

4.5 DESCOBERTA E INVOCAÇÃO DE AÇÕES

O modo de interação dos utilizadores com as ações é completamente distinto das aplicações tradicionais, no entanto esta não é a única diferença. Os utilizadores não necessitam de instalar nada para conseguir invocar uma determinada ação no assistente. Para ser possível utilizar uma determinada ação é apenas necessário que esta seja submetida e devidamente aprovada, só assim pertencerá ao repositório de ações - *Actions Directory* [4] - onde os utilizadores podem descobrir e desfrutar de novas ações para o seu assistente.

Vimos anteriormente que os utilizadores não podem descarregar as ações como fariam com aplicações tradicionais para os seus smartphones. Todavia, permitir que os utilizadores descubram uma ação é um factor muito importante para o sucesso a longo prazo, assim como é o modo como invocam a mesma. Deve ser preocupação do programador controlar estas variáveis definindo um nome e frases de invocação da sua ação. Utilizando o exemplo anterior da ação GeekNum exemplificaremos os diferentes métodos de invocação e descoberta de uma ação.

4.5.1 *Métodos de invocação*

- **Através do nome de invocação**: os utilizadores podem invocar explicitamente as ações - *Ok google, talk to GeekNum*.
- **Através do nome de invocação e uma frase para a ação**: os utilizadores podem invocar explicitamente as ações com uma frase descritiva do seu pedido em conjunto com o nome da ação - *Ok Google, talk to GeekNum to learn about the number 42*.

4 https://assistant.google.com/explore

4.5.2 *Métodos de descoberta*

- **Através de uma frase para a ação:** os utilizadores podem descobrir uma ação ao fazer diretamente um pedido - *Ok Google, learn about the number 42*. O assistente conhece uma ação que consegue de facto satisfazer este pedido do utilizador e aconselha-o - *For that, try saying 'Ok Google, let me talk to GeekNum'*.
- **Através da directoria de ações:** os utilizadores podem navegar pelo *Actions Directory*, um repositório contendo todos os projectos de ações submetidos e aprovados, disponíveis para uso a qualquer momento. Cada projecto contém informação detalhada das suas ações, os dispositivos que suporta, exemplos de invocações/pedidos, avaliações de outros utilizadores, etc.
- **Link para a ação:** é ainda possível gerar um link para o projecto de ações que pode posteriormente ser partilhado em qualquer plataforma. Ao carregar no link os utilizadores são redirecionados para o Google Assistant e a ação será logo de seguida invocada.

4.6 DIALOGFLOW

Vimos no Capítulo 3 que será necessário o assistente reconhecer as intenções do utilizador através da utilização de *Natural Language Understanding* (NLU). Para facilitar o processo de desenvolvimento desta camada, a Google oferece o Dialogflow, um serviço *web-based* que usa um agente para processar as frases do utilizador. Este serviço permite integrar aplicações de conversação com o assistente ou até com outras plataformas em que uma aplicação de conversação faça sentido, por exemplo o Facebook Messenger. Apesar desta ferramenta ser facultada, a Google não exige que seja utilizada, deixando para os programadores a opção de usar ou não a mesma. Quando esta não é utilizada e estes preferem ter as suas próprias soluções de NLU para criar ações, deve ser utilizado o Actions SDK[15].

4.6.1 *Integração com ações*

A criação de um agente no Dialogflow permite que este auxilie o processo de reconhecimento de intenções do utilizador e ainda de informação/dados a estas associados.

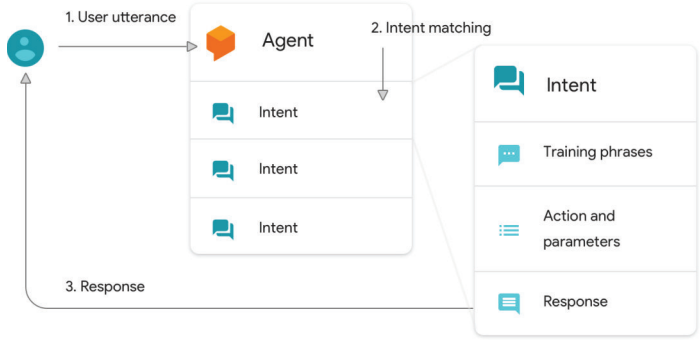


Figura 12: Funcionamento do Dialogflow perante uma fala de um utilizador. [14]

O diagrama da Figura 12 demonstra os três passos básicos que constituem o processo de receção de mensagens do utilizador e cálculo da resposta de acordo com estas. Assim que o agente recebe uma mensagem provinda do utilizador, tentará fazer correspondência desta com uma das intenções previamente estabelecidas. Estas intenções têm de ser construídas com base em frases exemplo, entidades relevantes e/ou eventos que as possam despertar - veremos mais adiante o papel de cada um dos componentes que constituem uma intenção. Assim que a correspondência de uma intenção é feita, a resposta contida nesta será retornada para o utilizador.

4.6.2 *Desenvolvimento de um agente*

Antes do desenvolvimento é necessária uma fase de especificação da aplicação, pensando nos diversos casos de uso da mesma, assim como diálogos exemplo que façam sentido dentro do domínio de cada um destes, tal como foi visto em 3.1.1.

Para definir ações no Dialogflow é necessário, numa fase inicial, criar intenções no agente de Dialogflow e atribuí-las como pontos de entrada para a nossa aplicação. Existem dois tipos de ações deste género:

- **Default Action:** qualquer projeto no Dialogflow deve ter apenas uma ação que é invocada quando os utilizadores a chamam pelo nome - *Ok Google, Talk to GeekNum*.
- **Intenções adicionais com links diretos para a conversa:** estas ações são desencadeadas quando os utilizadores invocam a ação através do nome em conjunto com uma frase de invocação específica.

Agora que os utilizadores têm forma de começar uma conversa com o assistente, devemos permitir que a conversa se estenda entre ambos. Normalmente um agente tem várias intenções que representam um conjunto de interações possíveis para o utilizador.

Quando um utilizador diz algo, o agente do Dialogflow tentará fazer correspondência da fala para uma intenção particular e só assim poderá retornar uma resposta de acordo com essa mesma intenção. Quando esta correspondência não acontece é possível definir *callbacks* para não deixar que o utilizador fique sem resposta. As falas do utilizador fazem correspondência com uma intenção através das frases exemplo (ou *training phrases*) e palavras importantes, ou outros valores que são definidas dentro destas.

4.6.3 *Intenções*

Para criar uma intenção (ou *intent*) é necessário ter em conta os componentes que as constituem, que permitem mapear o que o utilizador diz ao o que o agente irá responder.

- **Nome da intenção:** tal como indica é o nome da intenção, este nome é passado para o *fulfillment* como forma de identificação da intenção que fez correspondência com a invocação do utilizador.
- **Frases exemplo:** frases exemplo ou *training phrases* são exemplos de invocações que os utilizadores podem utilizar e que fazem correspondência a essa mesma intenção. Através de aprendizagem o agente irá fazer correspondência das frases dos utilizadores com frases idênticas às fornecidas como exemplo. Um exemplo de frases para treinar o modelo do agente poderiam ser *Teach me about the number 42* e *I want to know more about the number 13*.
- **Ação e parâmetros:** define como informação relevante (parâmetros) são extraídos das falas do utilizador. Exemplos deste tipo de informações incluem datas, tempos, nomes, sítios e ainda mais. É neste componente que são utilizadas as entidades com tipos de informação relevante para a intenção. Neste campo estaria presente, por exemplo, a entidade *number* devidamente identificada em cada uma das *training phrases* fornecidas.
- **Resposta:** resposta estática dada ao utilizador.
- **Fulfillment:** alternativa ao Resposta. Neste componente deve ser sinalizado se esta intenção deve ou não fazer um pedido a um serviço/aplicação para realizar algum tipo de lógica. Grande parte das intenções acabam por ter esta opção ativa já que é através da mesma que é possível construir respostas dinâmicas e/ou realizar qualquer tipo de processamento consoante o pedido do utilizador.

Opcionalmente, temos ainda:

- **Contextos:** contextos representam o estado atual do pedido do utilizador, já que permitem que seja transportada informação entre intenções. As informações arma-

zenadas através de contextos são importantes para, por exemplo, definir o fluxo de conversação.

- **Eventos:** os eventos permitem a invocação de intenções através de algo que tenha acontecido, ao invés de algo que o utilizador possa ter dito. O Dialogflow suporta eventos de diversas plataformas com base nas ações que os utilizadores têm nas mesmas.

4.6.4 *Entidades*

Para a definição de intenções, muitas vezes acaba por ser necessária a utilização de entidades (ou *entities*). As entidades são um mecanismo do Dialogflow para extrair dados úteis das invocações dos utilizadores.

Enquanto que as intenções permitem ao agente entender a motivação por detrás das invocações de um utilizador, as entidades são utilizadas para reconhecer informação de partes específicas dessas invocações - informações relativas a moradas, nomes de produtos, datas ou até quantidades segundo diferentes unidades. Qualquer tipo de dados que se pretenda extrair a partir do pedido do utilizador deve ter uma entidade correspondente. Existem três tipos de entidades distintos:

- **Entidades do sistema (*System entities*):** O Dialogflow oferece um conjunto numeroso de entidades que permitem aos agentes extrair informação sobre uma ampla variedade de conceitos. Exemplo disso são entidades para extrair informação sobre datas, tempo, localização, etc.

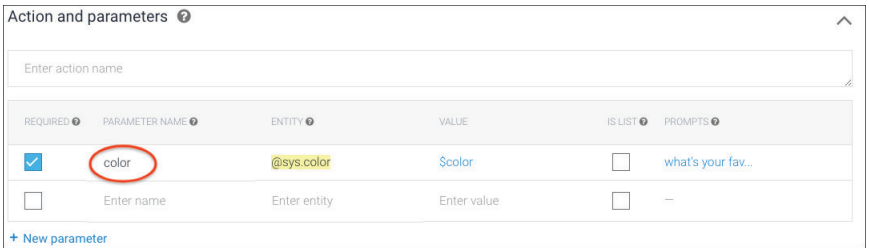


Figura 13: Exemplo da utilização de entidades do sistema.

- **Entidades do programador (*Developer entities*):** Por vezes é necessário extrair informação sobre conceitos para além dos que são oferecidos pelas entidades do sistema do Dialogflow. Um exemplo disso seria uma marca criar uma entidade deste tipo para reconhecer o conjunto de nomes dos seus produtos.

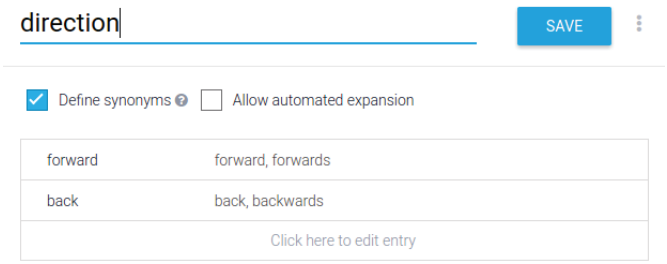


Figura 14: Exemplo da utilização de entidades do programador.

- **Entidades de sessão (*Session entities*):** É ainda possível definir entidades que se aplicam a uma conversa em específico. Estas entidades correspondem a um utilizador, em particular à sua sessão, e podem ser utilizadas durante a conversa deste com o assistente.

4.7 Componente de lógica de negócio

A introdução de processamento para os pedidos dos utilizadores obriga à criação de um componente onde a lógica de negócio da aplicação se irá inserir, também chamado de *fulfillment*. Este componente receberá pedidos provindos do agente do Dialogflow quando a resposta para um determinado pedido não possa ser calculada unicamente pelo próprio agente, isto é, quando não existe uma resposta estática para uma determinada intenção que tenha feito correspondência com a invocação do utilizador.

Apresentam-se, de seguida, exemplos de casos de uso onde a utilização de um *fulfillment* seria indispensável de modo a alargar o conjunto de funcionalidades oferecidas pelo agente:

- Gerar respostas dinâmicas com base em informações recolhidas de uma base de dados
- Realizar pedidos para comprar produtos que o utilizador pediu
- Implementar regras e condições para vencer ou não um jogo
- No caso do *GeekNum* poderia ser utilizado para fazer pedidos a um serviço externo para obter factos interessantes sobre um determinado número

A comunicação entre a plataforma *Actions on Google* e o *fulfillment* é através de HTTP/JSON. Desta forma o *fulfillment* deverá ser capaz de responder a pedidos HTTP provindos do *Actions on Google*. A Figura 15 retrata, de uma forma geral, as interações entre o Google Assistant, a plataforma Dialogflow e o *fulfillment* de uma ação.

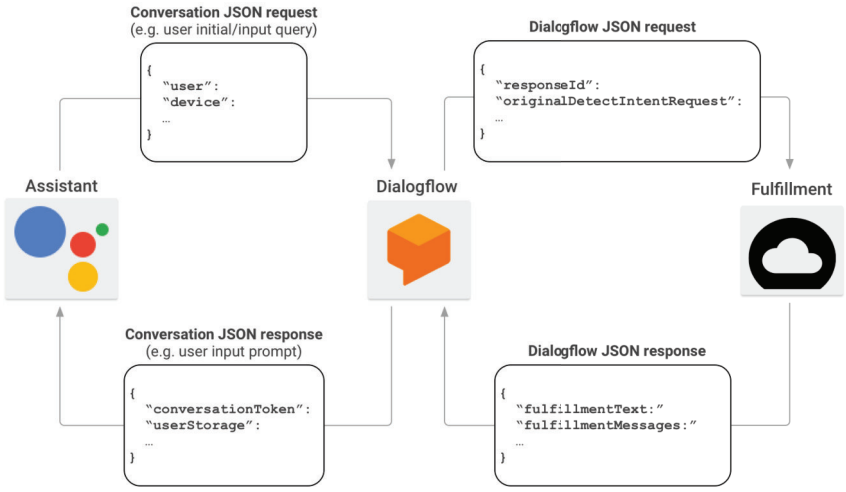


Figura 15: Invocações entre Google Assistant, Dialogflow e Fulfillment. [5]

Quando os utilizadores invocam a ação, o *fulfillment* receberá um pedido HTTP POST com formato JSON, em que o conteúdo deste descreve muito detalhadamente o pedido do utilizador. Neste pedido estão incluídas informações de extrema importância para facilitar o processamento do *fulfillment*, tais como as intenções e entidades identificadas pelo Dialogflow. O *fulfillment* tem a responsabilidade de atuar de acordo com estas informações, realizando o pedido do utilizador e posteriormente gerando uma resposta em formato JSON para lhe retribuir. A resposta gerada deve estar de acordo com padrões definidos para que seja possível o assistente fazer a tradução da mesma para voz ou outros tipos de media disponíveis para apresentar ao utilizador.

4.8 SMART HOME ACTIONS

Nesta secção abordaremos a construção de ações direcionadas para o controlo de dispositivos de IoT em busca da construção de uma *smart home*. Assim sendo, através da plataforma *Actions on Google* podemos construir *smart home actions*, que oferecem um objetivo mais concreto do que as tradicionais ações vistas anteriormente, já que permitem os utilizadores controlar dispositivos IoT através do Google Assistant. A construção destas ações permite aos utilizadores conectarem-se aos dispositivos e gerir os seus estados, através de uma infraestrutura existente na *cloud*.

Este tipo de ações é possível graças à existência da *Home Graph*, uma base de dados desenvolvida a pensar neste contexto de interação com dispositivos dentro de uma casa inteligente. Através desta base de dados é possível armazenar informação sobre a casa

5 Imagem retirada da documentação do *Actions on Google*.

como um todo e os dispositivos que nesta se inserem, o que permite consultas em real-time sobre o estado de cada dispositivo. É possível, por exemplo, armazenar informação sobre múltiplos dispositivos inteligentes (por exemplo luzes, televisão ou colunas) que um utilizador tenha disponível para controlo, mesmo que estes se encontrem em diferentes divisões da sua casa.

A estruturação e desenvolvimento de ações deste género é um pouco distintiva das ações tradicionais. No contexto destas ações será necessária a criação de apenas um **fulfillment**, servindo como serviço/servidor para lidar com os diferentes tipos de intenções recebidas, já que a forma como os utilizadores despertam ações e tradução dos pedidos destes em intenções deixa de ser necessária. Isto é, deixa de fazer sentido utilizar alternativas como o Dialogflow ou o Actions SDK para a camada de NLP, já que esta é completamente abstraída do programador, dado que é desenvolvida pela Google e os programadores não têm qualquer controlo sobre a mesma.

4.8.1 *Intenções*

Enquanto que para as ações tradicionais, ou conversacionais, o *fulfillment* deve lidar com os diferentes tipos de intenções definidas no módulo de NLP, para o correcto funcionamento das *smart home actions* deverá saber processar única e exclusivamente os seguintes tipos de intenções:

- **SYNC:** ativada quando o Google Assistant pretende reconhecer os dispositivos que um determinado utilizador tem. Desta forma, sempre que um utilizador se autentica esta intenção é recebida no servidor e este deve responder com a informação de todos os dispositivos conectados do utilizador.

- **QUERY:** uma intenção deste tipo é recebida no servidor quando o Google Assistant, como intermediário do pedido de um utilizador, pretende reconhecer o estado de um determinado dispositivo. O servidor deve reagir a esta intenção respondendo com informação detalhada acerca do estado do dispositivo.

- **EXECUTE:** despertada quando o Google Assistant, devido ao pedido de um utilizador, pretende controlar um dispositivo. O servidor para além de processar o pedido e alterar devidamente o estado do dispositivo deve responder com o estado resultante do dispositivo após o processamento.

- **DISCONNECT:** intenção recebida quando o utilizador pretende terminar sessão da ação, deixando de poder controlar os dispositivos que a esta estavam associados.

4.8.2 *Tipos de dispositivos*

Os dispositivos do utilizador devem ser de um determinado tipo, sendo que este tem normalmente associado um conjunto de funcionalidades recomendadas - no caso de um dispositivo do tipo *Light* podemos certamente esperar que uma das suas funcionalidades seja a possibilidade de o ligar/desligar. O tipo de dispositivo ajuda o módulo de *Natural Language Processing* do próprio assistente a identificar a intenção e entidades nas quais uma determinada mensagem do utilizador se deverá traduzir. Na Figura 16 são apresentados alguns dos tipos de dispositivos disponíveis no assistente.

Ainda para o exemplo de um dispositivo do tipo *Light*, o assistente deve esperar inputs/-mensagens do género:

- *Turn my light on please*
- *Turn off the light*
- *Turn on the light in my kitchen*
- *Turn my bedroom's light off*

Device	Device Type	Description	Recommended Traits
Camera	`action.devices.types.CAMERA`	Cameras are complex and features will vary significantly between vendors. Over time, cameras will acquire many traits and attributes describing specific capabilities, many of which may interact with the video/audio stream in special ways, such as sending a stream to another device, identifying what's in the stream, replaying feeds, etc As such, cameras also interact with other devices - especially screens and other media targets.	• CameraStream
Coffee maker	`action.devices.types.COFFEE_MAKER`	Interactions with coffee makers may include turning them on and off, adjusting the target temperature, and adjusting various mode settings.	• Modes • OnOff • TemperatureControl • Toggles
Dishwasher	`action.devices.types.DISHWASHER`	Dishwashers can have start and stop functionality independent from being on or off (some washers have separate power buttons, and some do not). Some can be paused and resumed while washing. Dishwashers also have various modes and each mode has its own related settings. These are specific to the dishwasher and are interpreted in a generalized form.	• Modes • OnOff • RunCycle • StartStop • Toggles
Door	`action.devices.types.DOOR`	Door can be opened and closed, potentially in more than one direction .	• OpenClose
Fan	`action.devices.types.FAN`	Fans can typically be turned on and off and have speed settings. Some fans may also have additional supported modes, such as fan direction/orientation (for example, a wall unit may have settings to adjust whether it blows up or down).	• FanSpeed • Modes • OnOff • Toggles

Figura 16: Alguns tipos de dispositivos disponíveis, retirado da documentação da Google Smart Home [15]

4.8.3 *Tipos de funcionalidades*

Além de um tipo, os dispositivos devem também ter funcionalidades associadas, também chamadas de *traits*. Cada tipo de dispositivo tem um conjunto de funcionalidades recomendadas associadas, no entanto, estas não são obrigatórias e podem ser adicionadas ou removidas as funcionalidades que o programador assim entender. As funcionalidades de um dispositivo são o que lhes confere utilidade para os utilizadores - um dispositivo do tipo *Light* não poderá ser ligado/desligado caso não tenha a funcionalidade On/Off. Tal como nos tipos de dispositivos, os tipos de funcionalidades permitem ao módulo de *Natural Language Processing* do assistente reconhecer a intenção e entidades que um determinada mensagem do utilizador contém. Na Figura 16 são apresentados alguns dos tipos de funcionalidades disponíveis.

Name	Device Trait	Description	Recommended Device Types
Brightness	`action.devices.traits.Brightness`	Absolute brightness setting is in a normalized range from 0 to 100 (individual lights may not support every point in the range based on their LED configuration).	Light
CameraStream	`action.devices.traits.CameraStream`	This trait belongs to devices which have the capability to stream video feeds to third party screens, Chromecast-connected screens or an Android phone. By and large, these are currently security cameras or baby cameras. But this would also apply to more complex devices which have a camera on them (for example, video-conferencing robotics/devices or a vacuum robot with a camera on it).	Camera
FanSpeed	`action.devices.traits.FanSpeed`	This trait belongs to devices that support setting the speed of a fan (that is, blowing air from the device at various levels, which may be part of an air conditioning or heating unit, or in a car), with settings such as low, medium, and high.	• Air conditioning unit • Air purifier • Fan
LockUnlock	`action.devices.traits.LockUnlock`	This trait belongs to any devices that support locking and unlocking, and/or reporting a locked state.	Any device that support locking and unlocking, and/or reporting locked state.
OnOff	`action.devices.traits.OnOff`	The basic on and off functionality for any device that has binary on and off, including plugs and switches as well as many future devices.	• Light • Outlet • Switch • Many other appliances
OpenClose	`action.devices.traits.OpenClose`	This trait belongs to devices that support opening and closing, and in some cases opening and closing partially or potentially in more than one direction. For example, some blinds may open either to the left or to the right. In some cases, opening certain devices may be a security sensitive action which can require two-factor authentication authentication. See Two-factor authentication.	Any device that supports opening and closing.
StartStop	`action.devices.traits.StartStop`	Starting and stopping a device serves a similar function to turning it on and off. Devices that inherit this trait function differently when turned on and when started. Unlike devices that simply have an on and off state, some devices that can start and stop are also able to pause while performing operation.	Any - mostly appliances and vacuums and other things that have specific activity behavior above and beyond power

Figura 17: Alguns tipos de funcionalidades disponíveis, retirado da documentação da Google Smart Home [15]

4.8.4 *Request SYNC*

Uma vez que a intenção de SYNC só é despoletada quando o utilizador se autentica na aplicação e ainda, relembrando que é esta intenção que devolve informação relevante sobre os dispositivos (nome, fabricante, tipo e funcionalidades que dispõem), se por alguma razão esta informação for externamente atualizada, para que dentro do contexto da aplicação estas alterações sejam verificadas seria necessário que o utilizador terminasse sessão na aplicação e a voltasse a iniciar. A funcionalidade de Request SYNC, disponível através da API do *Home Graph* permite assim que seja recebido um pedido traduzindo uma intenção de SYNC na aplicação e deverá ser utilizada em situações em que ou é atualizada informação sobre um determinado dispositivo do utilizador, ou é adicionado/removido um dos seus dispositivos.

4.8.5 *Report State*

A funcionalidade de *Report State* permite às aplicações reportarem pro-ativamente o estado dos dispositivos do utilizador, armazenando estes na *Home Graph*, base de dados da Google à qual o programador não tem acesso, podendo apenas desfrutar da sua API disponível.

Desta forma, garantimos que assim que o assistente necessitar do estado de um determinado dispositivo não será necessária uma intenção de QUERY para a aplicação do mesmo, já que o estado foi previamente reportado para o *Home Graph*. Considerando um exemplo em que um utilizador tem vários dispositivos, geridos por aplicações distintas, e invoca um comando que abrange todos estes dispositivos. Se este comando necessitar do estado atual do dispositivo será necessário invocar, pelo menos, intenções QUERY para cada uma das aplicações que controlam estes dispositivos. Ao tirar partido do *Report State* isto deixa de ser necessário porque garantimos que cada uma destas aplicações foi reportando o estado dos seus dispositivos para o *Home Graph*, que o assistente irá consultar para obter as informações pretendidas.

Seguem-se as situações em que uma aplicação deverá recorrer à funcionalidade de *Report State*, e por isso fazer uso da API do *Home Graph* para reportar o estado atual dos dispositivos, após a receção das seguintes intenções:

- **SYNC**: seja no momento de autenticação, ou quando é adicionado/removido um novo dispositivo - Request SYNC;
- **QUERY**: quando é obtido o estado atual do dispositivo, note-se que este pode ser sido alternado por um meio externo à aplicação (por exemplo um interruptor físico);
- **EXECUTE**: quando o estado do dispositivo é alterado.

5

APRESENTAÇÃO DO PROBLEMA

Como vimos anteriormente, o desenvolvimento deste tipo de aplicações acaba por se refletir num processo pouco sistematizado e industrializado. No contexto de *smart home* isto dá origem a uma falta de flexibilidade imposta aos utilizadores que tiram partido de aplicações nos assistentes digitais para controlar os dispositivos inteligentes das suas casas.

Vimos anteriormente que, para o caso do Google Assistant, as aplicações smart home têm frases previamente definidas que são usadas para treinar os modelos responsáveis pelo reconhecimento e tradução dos pedidos dos utilizadores em informação pronta a ser processada por estas aplicações. Note-se que o método de processamento de pedidos dos utilizadores destas aplicações é realizado através da correspondência com diferentes tipos de intenções que podem processar: Sync, Execute, Query e Disconnect, tal como vimos anteriormente na secção 4.8.1.

A existência de um modelo de NLP previamente treinado é, para os programadores, uma grande vantagem e reduz significativamente o esforço necessário para o desenvolvimento destas aplicações, já que estes só têm de se preocupar com o processamento das quatro intenções anteriores, efetuando a lógica para controlo dos dispositivos segundo estas e assumindo que todos os pedidos do utilizadores são devidamente traduzidos para a correspondente intenção, isto é, quando um utilizador pretende ligar um dispositivo, através da invocação - *Turn on my light* - a mesma resultará numa intenção de Execute com as devidas entidades acopladas (dispositivo, comando, etc). No entanto, esta abstração do módulo de NLP, sob o qual não é possível ter qualquer tipo de controlo, impõe fortes limitações no tipo de interação que os utilizadores podem ter com estas aplicações, não permitindo que estes utilizem invocações personalizadas para controlar os seus dispositivos.

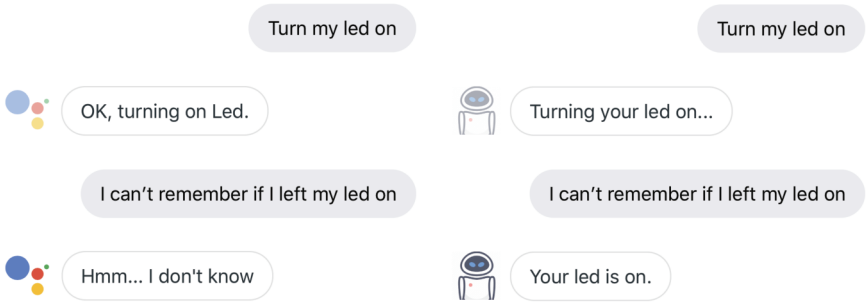


Figura 18: Comparação entre o comportamento atual para uma query customizada (esquerda) com o pretendido depois de implementada a funcionalidade (direita).

Para além disto, o desenvolvimento de aplicações para os assistentes digitais é ainda uma prática muito recente, sendo que as funcionalidades disponíveis para os utilizadores finais são muito reduzidas e limitadas. A procura de soluções para os assistentes digitais para acrescentar funcionalidades em falta fará também parte do âmbito da presente dissertação.

Um exemplo disto é o facto dos utilizadores não poderem perguntar ao Google Assistant quais as funcionalidades que um determinado dispositivo oferece para controlo, sendo que esta funcionalidade é uma mais-valia principalmente para utilizadores com muitos dispositivos inteligentes. Consideremos um utilizador com múltiplas lâmpadas inteligentes de dois fabricantes diferentes, A e B, distribuídas pela sua casa. Enquanto que as lâmpadas do fabricante A só permitem ser ligadas e desligadas, as lâmpadas do fabricante B permitem mudar a cor de iluminação. Nesta situação é impossível o utilizador identificar que uma lâmpada é do fabricante B, a não ser que tente mudar a cor de iluminação através de um comando por voz e este seja efetuado com sucesso. Outro exemplo são dispositivos que oferecem múltiplas funcionalidades, às vezes dezenas delas, não existe um comando disponível para os utilizadores conseguirem perceber todo o tipo de funcionalidades que podem tirar partido de um dispositivo específico.

A automatização e agendamento de ações para dispositivos inteligentes através do assistente está disponível através da utilização de Google Assistant Routines[1], no entanto a interação com esta funcionalidade só é possível através da aplicação mobile do Google Assistant, não podendo ser utilizada através da voz. Na Figura 19 podemos observar o comportamento atual do Google Assistant quando um utilizador pretende agendar um determinado comando comparando-o ao comportamento pretendido e esperado depois da implementação desta funcionalidade.

1 https://support.google.com/googlenest/answer/7029585

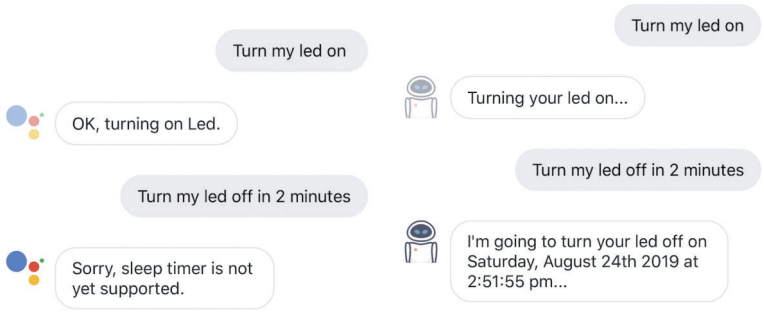


Figura 19: Comparação entre o agendamento de comandos atual (esquerda) com o pretendido depois de implementada a funcionalidade (direita).

Assim sendo, a criação de uma aplicação conversacional - aplicações que requerem desenvolvimento do módulo de NLP - que sirva de *middleware* entre as interações dos utilizadores e uma aplicação smart home, vem garantir não só mais flexibilidade na interação com aplicações para controlo de dispositivos inteligentes mas também uma maior facilidade no desenvolvimento de funcionalidades, dentro do domínio de IoT, que atualmente estão em falta no assistente.

5.1 DESAFIOS

A resolução dos problemas anteriormente descritos levanta inúmeros desafios que terão de ser devidamente explorados para cumprir com todos os requisitos na solução final a apresentar.

Na secção anterior vimos que para garantir flexibilidade nas invocações dos utilizadores é necessário o desenvolvimento de dois tipos de aplicações: conversacionais e smart home. Será então preciso explorar o desenvolvimento de cada um destes tipos de aplicações que, embora tirem partido da mesma infraestrutura, têm arquiteturas e uma abordagem de desenvolvimento muito distinta entre si.

A arquitetura destas aplicações, em particular das aplicações de smart home, que devem ser o principal foco, será um dos maiores desafios já que deve garantir escalabilidade horizontal para que, no mundo real, seja capaz de satisfazer pedidos de utilizadores sob milhares ou até milhões de dispositivos. Para além da arquitetura devem ser especificadas e desenvolvidas várias APIs na aplicação de smart home para possibilitar o acréscimo de novas funcionalidades ao assistente, como por exemplo uma API que permita obter as diversas funcionalidades que os dispositivos de um determinado utilizador oferecem. Para o contexto da dissertação, a aplicação conversacional que serve de middleware será o principal cliente destas APIs, no entanto estas devem ser genéricas permitindo ser utilizadas

por outro tipo de clientes, por exemplo para a criação de uma aplicação web que apresente uma dashboard com informações em real-time sobre os dispositivos de um determinado utilizador.

É de notar que a aplicação de smart home terá de efetivamente controlar dispositivos, isto é, se o utilizador executar um comando para ligar uma lâmpada, deverá ser possível visualizar isto fisicamente. Para isso será necessária a criação de uma aplicação adicional, que ficará a alojada num Raspberry Pi e que, de acordo com os pedidos recebidos, controle diversos periféricos a este conectados. Esta aplicação terá de estar disponível para o mundo exterior para que haja conectividade com a aplicação smart home.

A aplicação middleware ou conversacional deverá permitir os utilizadores controlarem os seus dispositivos não só com os comandos pré-definidos já disponíveis através do assistente, mas também com comandos personalizados. As funcionalidades acrescidas, como o agendamento de comandos sobre dispositivos ou pedidos sobre informação relativa a funcionalidades destes deverão também ser adicionadas a esta aplicação, cuja principal responsabilidade será traduzir as invocações dos utilizadores em pedidos à aplicação smart home. Na Figura 20 está representado o funcionamento entre as aplicações conversacional e smart home, em conjunto com os restantes componentes do sistema, tal como a aplicação no Raspberry Pi.

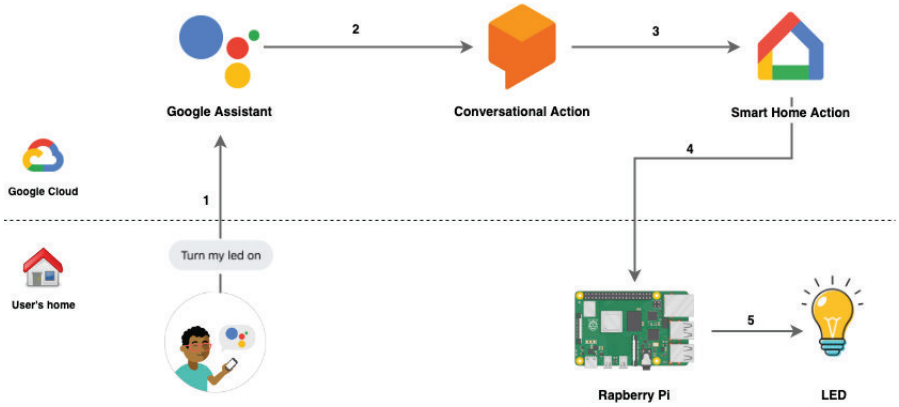


Figura 20: Funcionamento da *conversational action* e da *smart home action*.

O facto de tanto a aplicação conversacional como a de smart home terem de ser desenvolvidas na *cloud* constitui o desafio final, já que trará dificuldades tanto na configuração das mesmas como durante todo o seu processo de desenvolvimento, teste e debug.

6

ESPECIFICAÇÃO E MODELAÇÃO DO SISTEMA

Neste capítulo será detalhada a especificação e modelação do sistema, importante para o posterior desenvolvimento da arquitetura e implementação das aplicações. A especificação e modelação do sistema obriga a uma análise prévia sobre os diferentes desafios que surgem durante a fase de implementação, exigindo que sejam propostas à priori soluções para estes desafios.

Assim sendo, no presente capítulo começaremos com uma análise ao domínio do problema, identificando as principais entidades presentes e explicando os diferentes tipos de relacionamentos existentes entre estas. Com o domínio do problema apresentado, serão enumerados os diferentes casos de uso das aplicações, explicando brevemente em que consistem e fornecendo alguns exemplos de invocações que os utilizadores podem utilizar para tirar partido de cada um destes. Com base nestes, seguem-se a criação da persona e a definição do fluxo de diálogos que os utilizadores podem usar para interação com esta. Tanto a criação de uma persona como a definição dos diálogos da aplicação são características da especificação de aplicações de conversação, não sendo comuns na especificação das aplicações tradicionais a que estamos habituados. Por fim, é realizada a especificação da API a desenvolver, identificando os recursos e as rotas a implementar.

6.1 MODELO DE DOMÍNIO

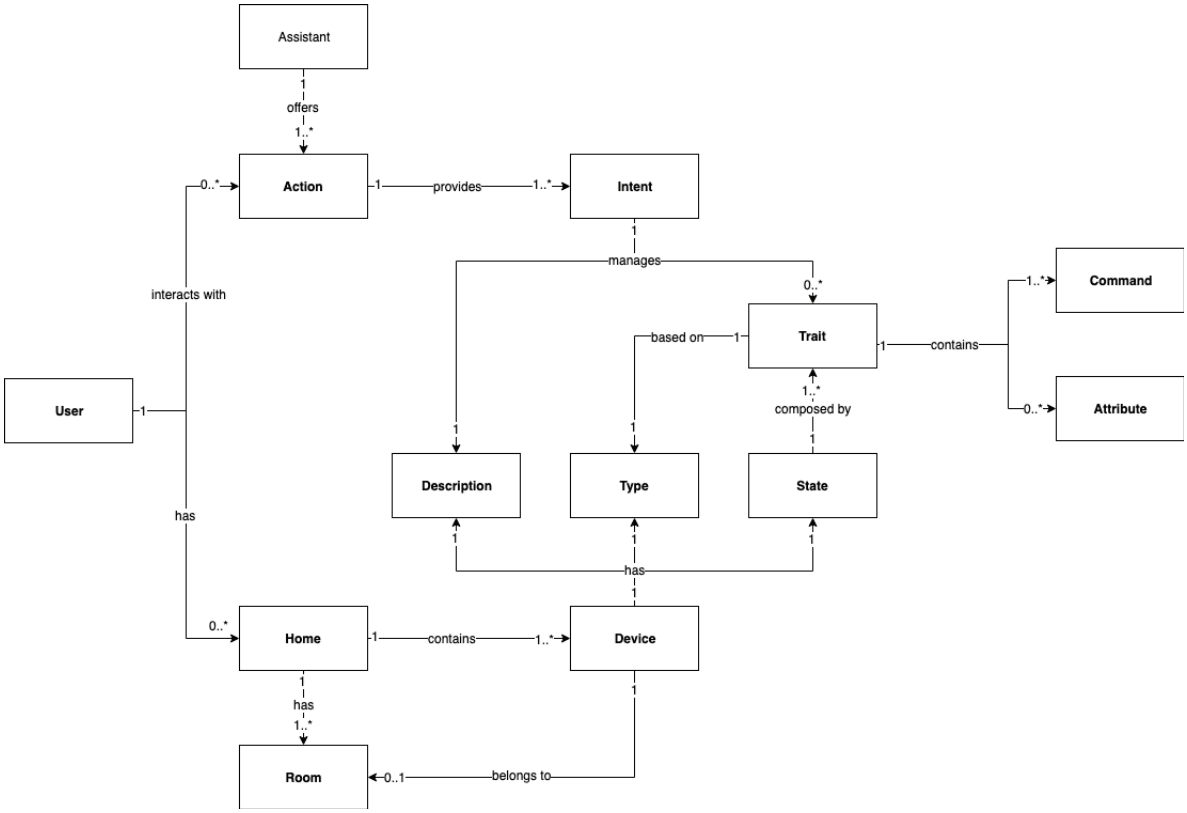


Figura 21: Modelo de domínio

User

Um utilizador do sistema deverá ser identificado pela sua conta Google já que a autenticação para as aplicações será realizada através desta. Como seria de esperar o utilizador deve ter uma casa, que por sua vez deve conter qualquer tipo de dispositivos IoT. Além disto, toda a interação com estes dispositivos será realizada através do Google Assistant, o que significa que este deverá ainda ter acesso a um dispositivo que permita a sua utilização - e.g. smartphone com aplicação do Google Assistant ou o próprio dispositivo Google Home.

Assistant

Esta entidade é a própria aplicação do Google Assistant, independentemente do dispositivo onde está embebida ou a partir do qual o utilizador fará uso da aplicação. Esta é a aplicação que serve de ponte para a comunicação do utilizador com um conjunto de actions.

Action

Como vimos anteriormente, serão aplicações para o assistente das quais o utilizador poderá tirar partido para realizar um conjunto diverso de funcionalidades. Neste caso particular esperamos que o utilizador faça uso de uma action para controlo e gestão dos dispositivos IoT presentes na sua casa. Estas operações serão traduzidas em intenções (intent) que após processadas devem refletir os objetivos do utilizador em ações sob os dispositivos. O utilizador deverá interagir com uma *Conversational Action* que manterá um estado de conversação com este e por sua vez traduzirá as suas intenções em pedidos a uma *Smart home Action* responsável por, de facto, processar estes pedidos satisfazendo as necessidades neles descritas, por exemplo ligar uma luz ou agendar o ato de ligar uma luz para uma determinada hora.

Intent

Esta será a entidade com toda a informação recolhida no pedido inicial do utilizador. Apesar de ser representada apenas por Intent, deverá incluir não só a intenção do utilizador como entidades com informação adicional necessária ao futuro processamento do pedido. A materialização do pedido inicial do utilizador, seja texto ou voz, numa intent será responsabilidade do NLP - neste caso o Dialogflow, tal como vimos em 4.6.

Home

A casa do utilizador será o contexto onde toda esta instrumentação deverá ocorrer. Uma casa deverá ter várias divisões e os dispositivos IoT distribuídos pelas respetivas.

Room

Esta entidade será importante para distinguir os dispositivos que se inserem numa determinada divisão da casa do utilizador, de modo a que possa facilitar tanto o reconhecimento como controlo dos dispositivos. No caso de existência do mesmo tipo de dispositivos distribuídos por diferentes secções da sua casa poderá ainda facilitar a sua identificação de modo a traduzir operações distintas nestes.

Device

Esta é entidade principal neste ecossistema. Representa cada um dos dispositivos que o utilizador pretende controlar. Cada dispositivo deverá conter três entidades que o caracterizam: uma descrição, um tipo e um estado, sendo este último caracterizado pelos Traits que o dispositivo disponibiliza, identificadores de ações que os utilizadores podem realizar.

Description

Esta entidade contém informações que descrevem e identificam o dispositivo, aliada a alguma meta-informação sobre o mesmo. Entre estas informações podemos contar com diferentes nicknames e sinónimos para o dispositivo, identificação do fabricante, traits que disponibiliza, etc.

Type

Cada dispositivo deve ser identificado com um tipo, de modo a facilitar a tradução dos pedidos do utilizador num determinado dispositivo, já que estas interações são baseadas no tipo de dispositivo que o utilizador pretende controlar. Exemplos de tipos de dispositivos foram já vistos anteriormente em 4.8.2.

State

Representa o estado que um dispositivo tem para um determinado trait. Isto é, para o trait OnOff, o dispositivo ou deverá estar on ou off. As intenções do utilizador deverão fazer correspondência com os traits do dispositivo que este pretende consultar ou controlar, para obter ou alterar informações do estado do dispositivo, respetivamente.

Trait

Representa uma funcionalidade que o dispositivo oferece para controlo por parte do utilizador. Os Traits estão, regra geral, associados ao tipo do dispositivo. Exemplos de traits foram já anteriormente apresentados em 4.8.3.

Command

Aliada a um trait, representa os comandos dos quais o utilizador pode tomar partido para interagir com o dispositivo. Usando como exemplo o trait StartStop teríamos os comandos de StartStop e ainda PauseUnpause.

Attribute

Tal como os comandos, aliam-se aos traits e identificam atributos que introduzem lógica sobre os comandos aplicados. Para o trait StartStop existe um atributo *pause* que, em caso negativo, impede os utilizadores de fazer Pause ao dispositivo.

6.2 SELECÇÃO DOS CASOS DE USO

Assumindo a realização de um prévio levantamento de requisitos para perceber quais as verdadeiras necessidades dos utilizadores, serão selecionados e descritos os casos de uso que realmente fazem sentido estar presentes na aplicação. Devido à situação particular em que a necessidade desta aplicação surge, isto é, o contexto da presente dissertação, o levantamento de requisitos não é um ponto indispensável para a definição dos casos de uso que devem ser garantidos pela aplicação a desenvolver.

Vimos anteriormente que o objetivo das aplicações a desenvolver é não só garantir flexibilidade aos utilizadores ao tirar partido das funcionalidades já presentes em aplicações smart home, mas também acrescentar algumas funcionalidades que venham melhorar a sua experiência ao controlar os seus dispositivos. Enumeram-se, se seguida, os diferentes casos de uso e exemplos de invocações possíveis para destes tirar partido.

1. **Consultar o estado de um dispositivo**
   - *Is the washer running?*
   - *Did I leave any light on in the kitchen?*
   - *Hey, I can't remember if I've already turned the AC on.*
2. **Alterar o estado de um dispositivo**
   - *Open the door, please.*
   - *Start recording with the outdoors camera.*
   - *I need light in my room!*
3. **Consultar as funcionalidades de um dispositivo**
   - *What traits does the washer have?*
   - *What can I do with the outdoors camera?*
4. **Agendar a alteração do estado de um dispositivo**
   - *Turn the fireplace on at 8 pm.*
   - *In 5 minutes, turn my bedroom lights off.*

   a) **Ser notificado quando o estado de um dispositivo é alterado**
      - *Your fireplace state changed.*
      - *Your bedroom lights state changed.*

5. **Limpar dados do utilizador (subscrições a notificações e dados da sessão de conversação atual)**
    - *Clear all my data, please.*

6.3 CRIAÇÃO DE UMA PERSONA

Vimos em 3.1.2 que a criação de uma persona é muito importante já que esta funciona como o front-end das aplicações de conversação. O fator usabilidade é em grande parte influenciado por esta, já que é importante que o modo de interação entre estas aplicações e os utilizadores seja consistente, garantindo-lhes uma experiência mais enriquecedora e coerente.

Para uma boa definição de uma persona, foram seguidos os passos recomendados nas Google Design Guidelines [17].

1. Uma pequena lista de adjetivos que descrevam a persona. O foco devem ser qualidades que sejam facilmente transmitidas para os utilizadores aquando utilização da aplicação.
2. Esta lista deve ser filtrada aos 4-6 adjetivos que melhor descrevam traços de personalidade da persona.
3. Enumerar diferentes personalidades que se encaixem dentro dos adjetivos anteriores.
4. Dentro das personagens anteriores escolher aquela que melhor se adequa para o propósito inicial da aplicação. Depois de selecionada, deve ser criada uma descrição que resuma a persona, de modo a que fique clara a maneira como diz ou age perante as interações do utilizador. O foco desta descrição devem ser traços de personalidade, com base nos adjetivos anteriormente identificados. Deve evitar-se especificar-se informações como género e/ou idade, já que estes nunca definem, de um modo crítico, a personalidade. Além disto, estas decisões podem dificultar a escolha de voz para a persona, por exemplo.
5. Encontrar ou até criar uma imagem/logo que represente visualmente a persona, o que permitirá aos utilizadores criarem uma memória visual sobre a mesma.
6. Seleção da voz adequada para a persona. Esta seleção deve ser entre uma voz sintetizada ou uma voz gravada.

Seguindo os passos anteriormente enumerados, segue-se a especificação de cada uma das etapas aplicadas para obter uma persona para a aplicação.

- **Nome**
  O nome escolhido para a persona foi *Dr. Smarthome*, já que o objetivo é ser alguém que demonstre conhecimento e seja uma referência dentro do contexto smart home.

- **Adjetivos-chave**
  - Geek
  - Prático
  - Experiente (em tecnologia)
  - Direto

  Os adjetivos-chave são do domínio da tecnologia, já que é exatamente o pretendido para garantir confiança aos utilizadores que tiram partido do mesmo para controlar os dispositivos inteligentes das suas casas.

- **Personagens que encarnaram os adjetivos**
  - Developer com experiência em automatização de dispositivos IoT
  - Robot programado para controlar dispositivos IoT

  Ambas as personagens se enquadram perfeitamente com as características anteriores. O foco é ser prático para o utilizador e demonstrar conhecimento no domínio da aplicação. Demonstrar experiência no domínio, é ainda um fator muito importante já que transmite confiança durante as conversas com o utilizador, característica presente tanto num developer com experiência em automatização de dispositivos como num robot programado para o efeito.

- **Descrição**

  Dr. Smarthome é um robot programado para ter interesse por tecnologia aplicada ao contexto smart home. Gosta de estar ao corrente de novos avanços tecnológicos dentro deste domínio e aplicar este conhecimento para ajudar os utilizadores que dele tiram partido. É notável o esforço que deposita em criar harmonia entre os diferentes dispositivos inteligentes dentro do mesmo contexto, de modo a agilizar as interações com estes por parte dos seus utilizadores. Quando questionado sobre o assunto, não gosta de se alongar dando respostas curtas e diretas.

Figura 22: Logo do Dr. Smarthome

- **Voz escolhida**

  A voz escolhida para a persona é a Male 2[1], já que trata a voz de homem, jovem e com a entoação semelhante à que os robots geralmente têm.

6.4 Escrita dos diálogos

Com os casos de uso bem definidos (por exemplo, a gestão do estado dos dispositivos) e a persona com que os utilizadores vão interagir devidamente apresentada, segue-se a definição dos diálogos que deverão ser disponibilizados. Os diálogos possíveis para interação com a aplicação deverão permitir a realização de cada um dos casos de uso apresentados em 6.2, mas também cumprir com as características previamente definidas para a persona da aplicação *Dr. Smarthome*, definidas em 6.3.

Apresenta-se de seguida o diagrama de fluxo dos diálogos possíveis (Figura 23), tanto os diálogos como anotações do diagrama estão em inglês, visto que será a linguagem utilizada nas conversas dos utilizadores com a aplicação. Apesar de abordar todos os casos de uso disponíveis, foram abstraídos do diagrama algumas situações menos comuns, como falta de input por parte do utilizador ou invocações não suportadas pela aplicação. O objetivo é conseguir retirar do diagrama o fluxo de diálogos para realizar cada uma das operações e as diferentes respostas dadas pela aplicação ao utilizador.

É de notar que apesar de no diagrama só ser apresentado um tipo de resposta, durante o desenvolvimento do módulo de NLP para a aplicação devem ser adicionadas respostas auxiliares para garantir uma melhor experiência de conversação ao utilizador, já que a aplicação deixa de usar sempre a mesma resposta para uma determinada situação.

A construção do diagrama foi baseada nas guidelines sugeridas pela própria Google para o efeito[17].

1 https://developers.google.com/actions/localization/languages-locales

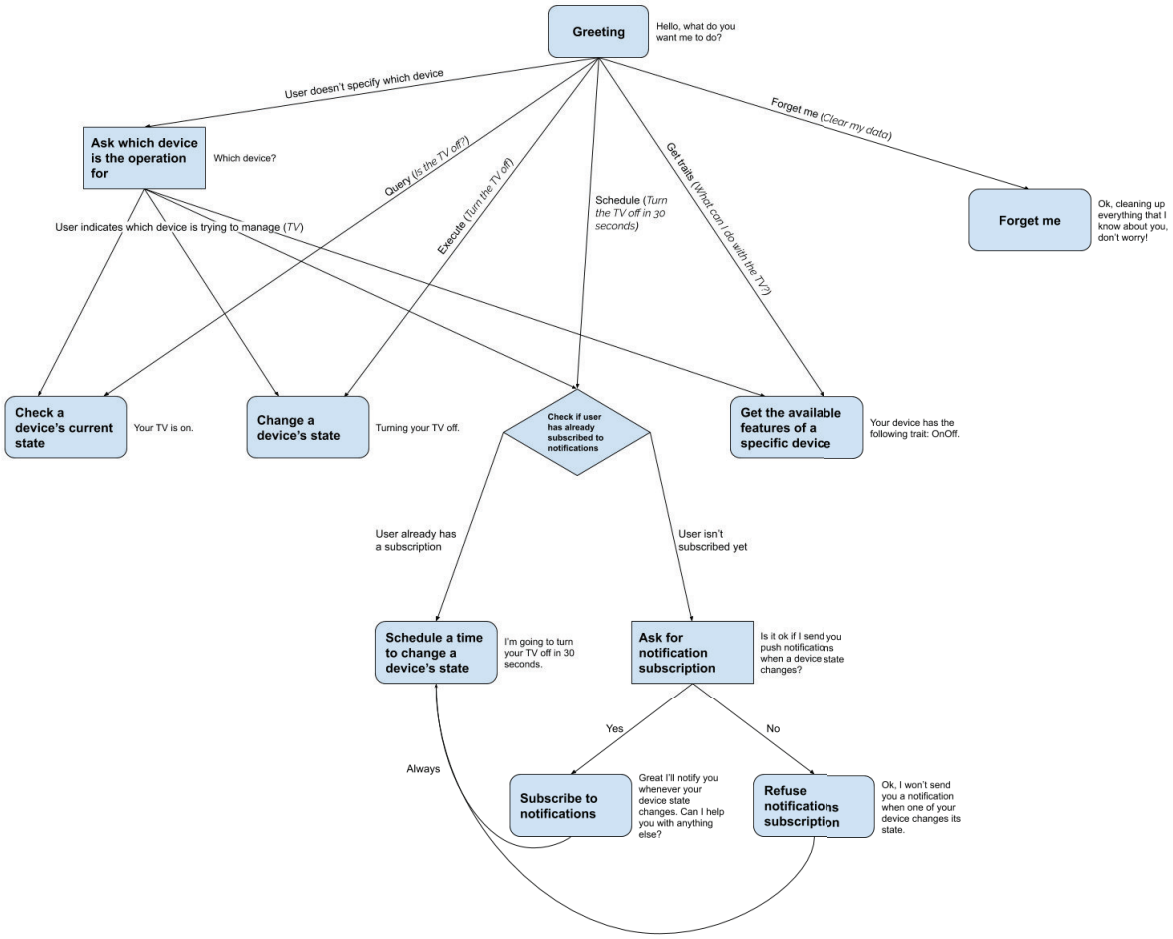


Figura 23: Diagrama do fluxo de diálogos

O ponto de entrada numa conversação da aplicação será através da saudação, onde é questionado ao utilizador o que é que este pretende que o *Dr. Smarthome* realize por si, demonstrando a sua capacidade de ser direto e prático para com os seus utilizadores. Das funcionalidades existentes, só uma delas é que não necessita que seja especificado um dispositivo para realizar uma determinada operação - *Forget me*. Esta funcionalidade é direta e serve exclusivamente para limpar os dados do utilizador relativos à subscrição de notificações ou a eventual informação que seja armazenada durante a sessão de conversação. As restantes funcionalidades necessitam de um dispositivo especificado para atuar. Se este for fornecido durante a invocação (e.g. *Is the TV off?*) a operação pode ser imediatamente realizada. No caso de não ser fornecido (e.g. *Is it Off?*) a aplicação deverá perguntar ao utilizador qual o dispositivo para o qual pretende realizar a operação (e.g. *Which device?*).

Vimos anteriormente que quando um utilizador agenda uma operação sob um dispositivo deve ser notificado assim que esta se realizar. Desta forma, quando o utilizador menciona que pretende agendar uma operação, a aplicação deverá verificar se o utilizador está subscrito a notificações, em caso positivo agenda imediatamente a operação (e.g. *I'm going to turn your TV on in 30 seconds*), em caso negativo pergunta-lhe se pretende subscrever às notificações (e.g. *Is it ok if I send you push notifications when a device state changes?*) e só posteriormente deverá ser feito o agendamento da operação.

É de notar que no diagrama o foco foi de uma única combinação funcionalidade/comando (ou trait/command), neste caso OnOff/OnOff, tal como podemos verificar pelos exemplos de invocações dos utilizadores e da própria aplicação (e.g. *Turn the TV off* e *Turning your TV off*), no entanto a aplicação deverá possibilitar a utilização de todos os tipos de funcionalidades e comandos disponíveis até ao momento, para permitir controlar dispositivos que façam uso destes. Como seria de esperar, a introdução de combinações adicionais pode exigir diferentes fluxos de diálogos para algumas das restantes funcionalidades.

6.5 especificação da api

Por fim, de modo a permitir a gestão dos dispositivos de cada utilizador é necessária a criação de uma API[2]. Esta API deverá ser disponibilizada pela aplicação de smart home e, mesmo sendo a aplicação conversacional o seu único cliente, deve ser pensada de modo a que no futuro possa ser explorada por diferentes clientes que, mesmo com objetivos diferentes, se insiram dentro do domínio smart home e/ou controlo de dispositivos inteligentes.

O foco da API desenvolvida deverá ser permitir gerir a informação dos utilizadores e dos dispositivos que cada um destes tem. A informação dos utilizadores deverá ser utilizada não só para os poder distinguir mas ainda para fazer a sua autenticação no sistema, enquanto que a dos seus dispositivos será utilizada durante a intenção de SYNC, que devolve informação sobre os dispositivos de um determinado utilizador, como vimos em 4.8.1. A informação para os dispositivos deverá conter, no mínimo, os dados destes utilizados pelas APIs da Google para controlo dos mesmos. Assim sendo, o tipo de dispositivo e o tipo de funcionalidades que oferecem deverão, certamente, estar presentes. O estado de cada um dos dispositivos, assim como dados relativos a subscrições das notificações não deverão fazer parte do contexto desta API.

O objetivo final deverá ser a disponibilização de uma API seguindo os princípios e restrições REST, de modo a que a solução apresente interfaces bem definidas que facilitem a comunicação com outras aplicações.

2 https://en.wikipedia.org/wiki/Application_programming_interface

6.5.1 *Formato*

O formato escolhido para utilização, refletindo o formato mais utilizado atualmente e tirando partido de todas as vantagens que com ele vêm acopladas é JSON (*JavaScript Object Notation*[3]). Este formato quando comparado a outras alternativas, sobressai-se em alguns pontos como a legibilidade, rapidez de serialização e a tradução praticamente imediata das estruturas de dados quando utilizado com linguagens como o JavaScript, sendo que esta será a linguagem utilizada para as aplicações a desenvolver.

6.5.2 *Recursos*

Iremos começar por apresentar os recursos e só posteriormente abordar cada uma das rotas disponibilizadas para fazer a gestão de cada um destes. As entidades principais presentes no modelo de domínio (6.1), o *Utilizador* e o *Dispositivo*, correspondem também aos dois recursos que a API oferece para gestão.

Utilizadores

O recurso de utilizadores terá informações básicas sobre os mesmos, de modo a que seja possível a sua identificação e ainda dados necessários para o sistema de autenticação do qual as aplicações farão uso, já que só o utilizador que possui um dispositivo deverá poder controlar as informações deste.

- **Id**: identificador único para cada utilizador;
- **Nome**: nome do utilizador;
- **E-mail**: e-mail do utilizador, corresponde ao e-mail da conta Google que está a ser usada para controlar os dispositivos;
- **Access token**: token gerado que deve ser utilizado para efetuar a autenticação do utilizador.

3 https://www.json.org/

```
{
    "id": "1",
    "name": "Dinis Peixoto",
    "email": "dinispeixoto96@gmail.com",
    "access_token": "fake_access_token"
}
```

Listing 1: Exemplo de um utilizador.

Dispositivos

Por outro lado, o recurso dos dispositivos oferece informação relevante para o seu controlo. É de notar que o estado dos dispositivos não será disponibilizado através da API, mas sim todas as informações que permitem que isto aconteça. Assim sendo, informações como o tipo de dispositivo e o tipo de funcionalidades que o dispositivo oferece estarão disponíveis através da API. Este tipo de informações é utilizado durante o pedido de SYNC da smart home action.

- **Id**: identificador único para cada dispositivo de um utilizador;
- **Nome**: objecto com diferentes informações acerca do nome do dispositivo, como o nome default ou até nicknames que o utilizador pode utilizar para controlar o seu estado;
- **Tipo**: tipo de dispositivo, vistos em 4.8.2
- **Funcionalidades**: ou também conhecidas como *traits*, tal como apresentado em 4.8.3. Em conjunto com o tipo de dispositivos, as funcionalidades definem as invocações que o módulo de NLP disponibilizará para controlar o estado deste. O dispositivo deverá ter um ou mais estados para cada uma das suas funcionalidades.
- **Informação adicional**: meta informação sobre o dispositivo. Aqui podemos encontrar dados sobre o fabricante, versões do hardware e software e ainda o modelo em específico.

```
{
    "id": "led",
    "name": {
        "defaultNames": [
            "My Led"
        ],
        "name": "Led",
        "nicknames": [
            "Led",
            "light",
            "led"
        ]
    },
    "type": "action.devices.types.LIGHT",
    "traits": [
        "action.devices.traits.OnOff"
    ],
    "deviceInfo": {
        "hwVersion": "1.0",
        "manufacturer": "Dinis Peixoto Raspberry Pi",
        "model": "led",
        "swVersion": "1.0.1"
    }
}
```

Listing 2: Exemplo de um dispositivo.

6.5.3 *Rotas*

Com os recursos e o conteúdo destes definidos, seguem-se então as rotas que deverão ser disponibilizadas para realizar operações sobre estes recursos.

Utilizadores

As rotas dos utilizadores definem as operações básicas CRUD (Create, Read, Update e Delete) comuns nas APIs REST. A rota para obter a lista de todos os utilizadores disponíveis é a única onde o *id* do utilizador não é obrigatório, correspondendo assim à única operação que não atua diretamente com uma única entidade. É de notar ainda que o único *status code* disponível será o 200, sendo que no caso de não existir qualquer utilizador esta operação devolverá uma lista vazia e não um erro em específico. O mesmo não acontece quando tentamos consultar, atualizar ou até eliminar um utilizador, usando como input da rota um *id* inexistente, já que isto resultará no *status code* 404, identificando que o recurso não

está disponível. Esta situação pode indicar, por exemplo, que o utilizador ainda precisa de ser criado. No caso de tentarmos criar um utilizador com um *id* que já se encontra em utilização, o *status code* resultante será 400. Qualquer tipo de exceções adicionais geradas durante o processamento de um pedido deverão resultar numa resposta com *status code* 500, identificando devidamente o erro que deu origem à exceção.

Método HTTP	**Rota**	**Descrição**	**Status code**
GET	/api/users	Obter a lista de utilizadores	- 200 (OK)
GET	/api/users/id	Obter a informação de um utilizador	- 200 (OK) - 404 (Not found)
POST	/api/users/id	Adicionar um utilizador do sistema	- 201 (Created) - 400 (Bad Request)
PUT	/api/users/id	Atualizar um utilizador do sistema	- 204 (No Content) - 404 (Not found)
DELETE	/api/users/id	Apagar um utilizador do sistema	- 204 (No Content) - 404 (Not found)

Tabela 1: Rotas para controlo de utilizadores.

Dispositivos

As rotas disponíveis para os dispositivos são muito semelhantes às dos utilizadores. Os dispositivos presentes no sistema estão diretamente associados a um utilizador, desta forma nas suas rotas será sempre necessário especificar o *id* do utilizador para o qual é pretendido gerir os seus dispositivos. Os *status code* de sucesso e erro para operações de consulta, criação, atualização ou remoção são idênticos aos apresentados para as rotas dos utilizadores. Apesar de não estar presente na tabela, qualquer operação sob o dispositivo de um utilizador cujo o *id* seja inválido resultará num *status code* 404, já que é impossível realizar operações em dispositivos de um utilizador inexistente. Assim como para as rotas dos utilizadores, quaisquer tipo de exceções adicionais geradas durante o processamento de um pedido deverão resultar num *status code* 500.

Método HTTP	Rota	Descrição	Status code
GET	/api/users/id/devices	Obter a lista de dispositivos do utilizador	- 200 (OK)
GET	/api/users/id/devices/device_id	Obter a informação de um dispositivo do utilizador	- 200 (OK) - 404 (Not found)
POST	/api/users/id/devices/device_id	Adicionar um dispositivo ao utilizador	- 201 (Created) - 400 (Bad Request)
PUT	/api/users/id/devices/device_id	Atualizar um dispositivo do utilizador	- 204 (No Content) - 404 (Not found)
DELETE	/api/users/id/devices/device_id	Apagar um dispositivo do utilizador	- 204 (No Content) - 404 (Not found)

Tabela 2: Rotas para controlo de dispositivos.

6.5.4 *Considerações*

A API apresentada tem como objetivo ser simples e permitir flexibilidade para futuras alterações, respeitando as diretrizes básicas que uma API REST apresenta. No entanto, esta não é, de todo, o foco da presente dissertação, o que levou a que algumas considerações adicionais fossem descartadas por não trazer qualquer valor acrescentado.

Um exemplo disto é o facto da API não apresentar a versão nas suas rotas (por exemplo /api/v1/users/), enquanto que durante o desenvolvimento destas aplicações o contracto exposto não vai ser alterado, isto é uma boa prática que se deve ter sempre em conta para que, no futuro, seja possível fazer alterações ao contracto sem ter um impacto muito significativo nos seus clientes. A exposição da versão nas rotas permite que diferentes clientes utilizem versões diferentes da API.

A falta de filtros e possibilidade de ordenação podia também ser um problema para o futuro da API. Na versão apresentada tem de ser o cliente a aplicar qualquer tipo de ordenação ou filtros, sendo obrigado a consultar toda a lista de entidades para a qual pretende obter algum tipo de informação. Ainda sobre esta propriedade o facto dos resultados não serem paginados para obter a lista de utilizadores ou dispositivos seria também uma alteração obrigatória, já que a performance dos pedidos a estas coleções iria diminuir consideravelmente com o aumento de entidades presentes nas mesmas.

Enquanto que para os casos de uso da aplicação cliente a desenvolver não era necessário, a criação de mais recursos garantia maior flexibilidade ao realizar operações nos dispositivos dos utilizadores. Um recurso adicional poderia ser o próprio nome, ou a meta-

informação desta, já que, como vimos anteriormente, corresponde a um objeto JSON independente da restante informação apresentada.

Por fim, a utilização dos cabeçalhos HTTP para fornecer maior interoperabilidade da aplicação que dispõem a API para com os seus clientes: disponibilização de um cabeçalho para o formato de serialização, ou até mesmo localização de um recurso criado através do cabeçalho de *location*[4] (deixaria de ser necessário o uso do *id* aquando criação de um recurso - método POST), seriam dois excelentes exemplos disso.

4 https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location

7

ARQUITETURA

Com a especificação e modelação do sistema concluída, iremos de seguida delinear as arquiteturas idealizadas. O capítulo começará com a apresentação de uma arquitetura geral do sistema e esta será posteriormente explorada mais detalhadamente ao nível das aplicações que o compõem. Durante este capítulo serão ainda feitas referências a algumas considerações sobre a infraestrutura utilizada já que esta determina a definição da arquitetura do sistema utilizada. Conceitos arquiteturais que foram explorados para garantir uma arquitetura flexível e escalável serão também mencionados, explicando para cada um destes de que forma contribuem positivamente para os objetivos da solução proposta. Será ainda feita referência aos desafios associados ao desenvolvimento destas arquiteturas fazendo a comparação destes com os desafios geralmente encontrados no desenvolvimento de arquiteturas para as aplicações tradicionais.

O desenvolvimento de uma arquitetura escalável deve ser uma das prioridades das aplicações a desenvolver. No entanto, devemos primeiro entender qual é a verdadeira necessidade para tal. Uma aplicação para gestão e controlo de dispositivos inteligentes deve possibilitar a sua utilização a milhares de utilizadores, sendo que este número é facilmente alcançado se tivermos em conta o constante crescimento da oferta e acessibilidade a estes dispositivos no mercado.

A cada dia que passa aumenta a quantidade e diversidade de dispositivos inteligentes disponíveis, por consequência os preços aplicados começam a reduzir drasticamente ficando assim mais acessíveis para os utilizadores. Se para milhares de utilizadores esperarmos que cada contenha não um mas vários destes dispositivos, facilmente chegaremos a valores na casa dos milhões para a quantidade de dados que precisam de ser controlados concorrentemente.

Deste modo, é necessário ter uma arquitetura que, independentemente do número inicial de utilizadores e/ou dispositivos, consiga escalar sem em que isso influencie o seu propósito inicial.

7.1 ARQUITETURA GERAL DO SISTEMA

Com base na Figura 20, foi explorada a interação entre cada uma das aplicações a desenvolver, em conjunto com informações acerca da infraestrutura da qual irão tirar partido, representado pelo diagram da Figura 24. A ideia será refletir como cada uma das aplicações interage no sistema, de modo a garantir que os casos de uso anteriormente apresentados estejam disponíveis para os utilizadores.

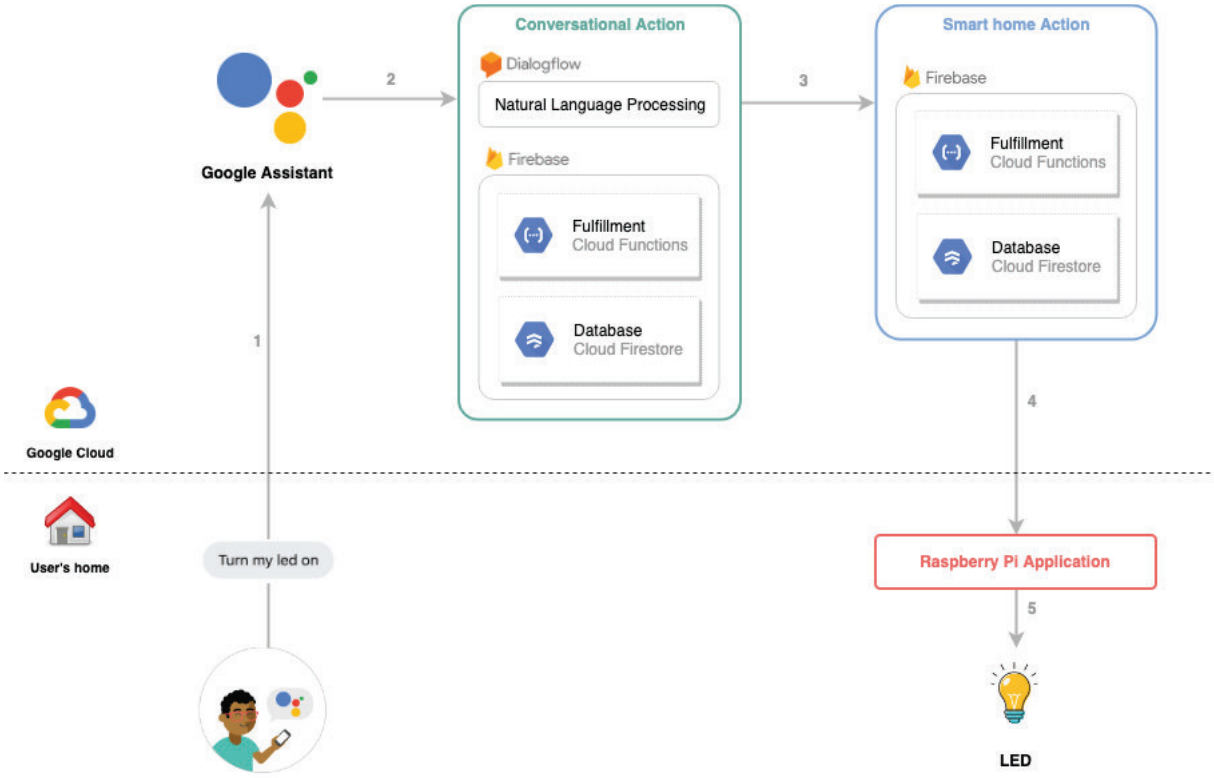


Figura 24: Diagrama da arquitetura geral do sistema.

1. O utilizador utiliza uma invocação para realizar uma operação sob um dos seus dispositivos. No exemplo do diagrama a frase utilizada é *Turn my led on*, embora pudesse ser qualquer frase que o utilizador entendesse, sendo que para isso seria apenas necessário treinar o modelo de *Natural Language Processing* da aplicação conversacional para o efeito. As invocações do utilizador são dirigidas para o Google Assistant.

2. O Google Assistant sabe, através do repositório de actions, como comunicar com a action que o utilizador pretende utilizar. A comunicação entre a aplicação do Assistant e as actions é realizada através de pedidos HTTP, tanto com o módulo de NLP (neste caso o Dialogflow) como com o *fulfillment* destas. Como sabemos, o utilizador estará a tirar partido de funcionalidades da action *Dr. Smarthome*, representada

no diagrama como *Conversational Action*. Esta aplicação deverá possuir um módulo de NLP, utilizando a plataforma Dialogflow para fazer a tradução dos pedidos dos utilizadores em intenções e entidades, sendo que esta informação será transmitida para o *fulfillment* da aplicação onde será realizada a lógica necessária. O *fulfillment* da aplicação será desenvolvido utilizando a plataforma *Google Cloud Functions*, que permite o desenvolvimento de funções que escalam individualmente, isto será discutido com mais detalhe em 7.2. A lógica da aplicação irá necessitar de uma base de dados, onde serão armazenadas informações sobre as subscrições dos utilizadores para as *push notifications*.

3. É de notar que a *Conversational Action* não altera, na realidade, o estado dos dispositivos, apenas faz um pedido à *Smart home Action* para realizar esta operação. Esta aplicação, por outro lado, não apresenta um módulo de NLP. Vimos anteriormente em 4.8 que existem intenções pré-definidas que terão de ser processadas. Para fazer o processamento destas intenções temos um *fulfillment* e uma base de dados, de modo similar à aplicação anterior, embora com responsabilidades distintas, já que aqui a lógica e base de dados existentes serão para controlo do estado e gestão de informação dos dispositivos, isto é, nesta base de dados estarão presentes, para cada utilizador, o estado e informações relativas a cada um dos seus dispositivos.

4. Para realizar operações sob dispositivos precisamos ainda de uma outra aplicação que tenha uma ligação direta aos dispositivos físicos. Esta estará em execução num *Raspberry Pi* para o efeito da presente dissertação e deverá receber pedidos provindos da aplicação smart home, traduzindo-os em comandos sob os dispositivos físicos.

5. Desta forma, no exemplo do diagrama, se o utilizador pretender ligar o seu LED, a aplicação presente no Raspberry Pi ao receber um pedido da aplicação de smart home deverá fazer a gestão dos seus periféricos para ligar efetivamente o LED.

7.1.1 *Infraestrutura Google*

As aplicações desenvolvidas para os assistentes digitais têm de estar disponíveis na *cloud* já que seria impraticável para os utilizadores ter de instalar cada aplicação no dispositivo físico que estão a utilizar para interagir com o Google Assistant. Ao desenvolver as aplicações para o Google Assistant, apesar de não ser estritamente obrigatório optou-se por utilizar as plataformas disponíveis no **Google Cloud**, facilitando a integração das diferentes aplicações entre si, mas também com o próprio Google Assistant.

A aplicação conversacional precisa de um módulo NLP e para desenvolvimento deste a Google disponibiliza a plataforma **Dialogflow**, onde os diálogos com os utilizadores para os

diferentes fluxos de conversação são especificados. Este é o ponto de entrada da aplicação, sendo que a informação recolhida nesta plataforma será transmitida para o *fulfillment* da aplicação, onde se encontra a lógica. O funcionamento do Dialogflow foi já explicado com detalhe em 4.6.

7.1.2 *Firebase*

Na infraestrutura da Google está disponível a plataforma Firebase[1]. Esta plataforma oferece inúmeras ferramentas para fazer gestão de aplicações numa infraestrutura na *cloud*. A utilização do Firebase releva-se muito importante para o desenvolvimento das aplicações já que o desenvolvimento deste tipo de aplicações tem de ser realizado na *cloud*, implicando desafios acrescidos. Para testar as aplicações será necessário fazer deploy das mesmas para a infraestrutura, consulta de *logs*, deteção de erros, conectividade entre aplicações e afins, serão alguns dos use-cases para os quais esta plataforma será utilizada.

As ferramentas, dentro da plataforma Firebase, indispensáveis ao desenvolvimento das aplicações serão o **Cloud Functions** e o **Cloud Firestore**, que serão de seguida explicados com maior detalhe. De notar que ferramentas de monitorização, instrumentação e qualidade, embora tenham sido utilizadas não serão mencionadas visto que servem apenas como auxílio ao desenvolvimento.

7.1.3 *Cloud Functions*

A *Google Cloud Functions* oferece um ambiente para execução de aplicações *serverless* (veremos mais adiante em 7.2). As *Cloud Functions* são nada mais do que funções simples, com um propósito único que são executadas quando um determinado evento (*cloud event*[2]), ao qual estão relacionadas, acontece. Deste modo, quando um evento que está a ser observado é espoletado, a função correspondente inicia a execução. Não há necessidade de preocupações com gestão da infraestrutura e/ou servidores necessários para alojamento destas funções, visto que isto é responsabilidade da própria plataforma onde estas se inserem.

1 https://firebase.google.com/
2 https://cloud.google.com/functions/docs/concepts/events-triggers

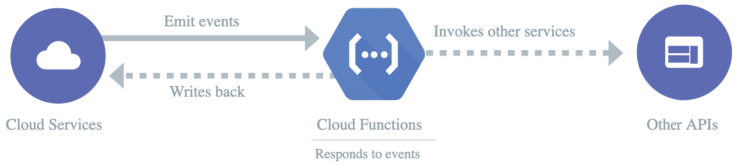


Figura 25: Funcionamento das Cloud Functions [16]

Existem múltiplos tipos de eventos na *cloud* aos quais as *Cloud Functions* podem reagir. Estes eventos podem ser alterações nos dados de uma base de dados, ficheiros adicionados ao *filesystem* ou até a criação de uma nova instância de uma máquina virtual. Para estes e muitos outros tipos de eventos é possível criar funções que reagem ao seu acontecimento executando uma determinada ação. O tipo de *Cloud Functions* que iremos desenvolver são *Webhooks*[3], ou seja funções que irão reagir a pedidos HTTP.

Neste momento a utilização das *Cloud Functions* está disponível para três *runtime environments*, sendo estes Node.js, Python 3 e Go, relembrando que a linguagem a utilizar para a implementação das aplicações será JavaScript e o *runtime environment* Node.js.

7.1.4 *Cloud Firestore*

A *Cloud Firestore* é uma base de dados NoSQL que, tal como as *Cloud Functions*, está alojada na infraestrutura da Google. Neste momento é possível interagir com esta base de dados através de SDKs para Node.js, Python, Go e Java, ou até através de uma API REST ou RPC.

Os dados armazenados na *Cloud Firestore* seguem um padrão semelhante ao usado em MongoDB[4]. Os dados são guardados em documentos que contêm campos mapeados para valores. Por sua vez, estes documentos são agrupados em coleções, que permitem uma forma de organização dos documentos, facilitando a aplicação de futuras queries nestes - tal como se pode ver na Figura 26. Este modelo de dados permite ainda que sejam criadas subcoleções dentro dos documentos para criar estruturas de dados mais complexas e flexíveis.

3 https://en.wikipedia.org/wiki/Webhook
4 https://docs.mongodb.com/manual/core/data-modeling-introduction/

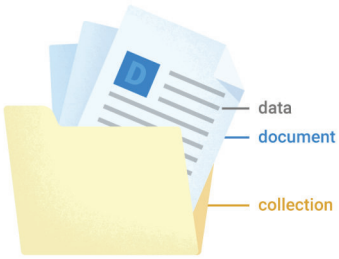


Figura 26: Modelo de dados utilizado na Cloud Firestore [13]

A natureza NoSQL aliada ao modelo de dados utilizada na *Cloud Firestore* permite que as queries realizadas neste tenham elevado desempenho e que seja facilmente escalável já que este processo é completamente automático e fica ao cargo da infraestrutura da Google.

7.2 Arquitetura Serverless

Arquitetura *Serverless*, também conhecida como *Serverless computing* ou FaaS (*Function as a service*) é um *design pattern* onde as aplicações estão alojadas num serviço de terceiros, eliminando a necessidade de gestão, por parte dos programadores, de tanto o servidor como o próprio hardware onde da qual a aplicação tira partido.

Numa arquitetura *serverless* as aplicações são desenvolvidas como funções que podem ser invocadas e escaladas individualmente, sendo que estas funções são os pontos de entrada da própria aplicação. O facto de serem desenvolvidas como funções independentes permite fugir ao conceito tradicional de servidores *always-on*, onde temos servidores continuamente disponíveis para receber pedidos, implicando, regra geral, elevados custos operacionais. Atualmente as três maiores empresas que oferecem serviços na *cloud* (Google, Amazon e Microsoft) já têm a sua própria solução do género para oferecer aos programadores[28]. As aplicações para os assistentes encaixam perfeitamente nos casos de uso desta arquitetura já que são compostas por intenções, sendo que estas são autónomas e independentes entre si, ou seja, cada intenção pode corresponder a uma destas funções.

Quando se pretende alojar uma aplicação para estar disponível para os utilizadores, normalmente é necessário fazer a gestão de um servidor físico ou virtual, um sistema operativo e ainda um servidor web responsável pelo ambiente onde a aplicação ficará em execução. Com a utilização de serviços na *cloud* como IaaS (*Infrastructure as a service*) é possível abstrair algumas destas preocupações ao nível do hardware, no entanto é igualmente necessária a gestão do software utilizado, tanto ao nível do sistema operativo como do próprio servidor web.

Por outro lado a utilização de serviços como PaaS (*Platform as a service*), como o Heroku[5], oferecem alguns benefícios relativamente ao IaaS já que abstraem o programador da gestão mesmo ao nível do software utilizado onde as suas aplicação se vão alojar. Basicamente os programadores fazem uma configuração mínima, especificando detalhes sobre a aplicação que vão alojar na *cloud*. Em parte o PaaS é muito semelhante ao FaaS, ambos oferecem benefícios ao nível da gestão da plataforma, seja software ou hardware, onde as aplicações ficam alojadas. No entanto existe uma diferença muito importante entre estes.

Com a utilização de um PaaS, é feito *deploy* das aplicações como uma unidade e são desenvolvidas da forma tradicional, através da utilização de uma *framework web*, seja ASP.NET, Flask ou Ruby on Rails. A escalabilidade para estas aplicações é definida ao nível da própria aplicação como um todo.

Já com a utilização de um FaaS a aplicação é particionada em funções individuais, independentes e autónomas. Cada uma destas funções fica alojada no serviço e é automaticamente escalável pelo fornecedor do serviço, isto é, se uma determinada função estiver a receber consideravelmente mais pedidos que outra serão dedicados mais recursos a esta para que não haja qualquer tipo de degradação da sua performance. Este fator torna-se fulcral para uma melhor gestão dos custos operacionais, já que só se estará a pagar os recursos que estão efetivamente a ser utilizados. Desta forma, com uma arquitetura *serverless*, o foco é nas funções individuais que constituem uma determinada aplicação.

Até então o foco foi explicar o que era uma arquitetura *serverless*, dando um pouco de contexto quando comparada a outras alternativas como IaaS ou PaaS. De seguida serão descritos, com detalhe, os benefícios para os programadores que a utilização de uma arquitetura *serverless* pode trazer.

- **Abstração da gestão tanto de software como hardware**

  O facto dos programadores serem completamente abstraídos da gestão ao nível de componentes de hardware e software que constituem a plataforma onde as aplicações se inserem permite que o foco seja exclusivamente direcionado para o desenvolvimento das funções que constituem as aplicações. A configuração necessária para fazer o *deploy* deste tipo de aplicações é também mínima, geralmente a identificação do ambiente de execução ou *runtime environment* das funções é suficiente.

- **Escalabilidade**

  Este tipo de abordagem garante mais escalabilidade quando comparado às aplicações tradicionais. A gestão de alocação de recursos para cada função é automática e garantida pelo fornecedor do serviço, isto é, se uma função receber mais pedidos

5 https://www.heroku.com/

pode escalar independentemente das restantes. Poderão ser alocados mais recursos para a mesma sem trazer qualquer impacto nas restantes funções. Deixa de existir a preocupação com quantos pedidos pode efetivamente uma aplicação processar em simultâneo, já que isto é, logo à partida, garantido pelo fator de *autoscaling*[18]. Para além disso, as aplicações passam também a ser mais modulares e seria possível ter, por exemplo, duas equipas de desenvolvimento diferentes a trabalhar na mesma aplicação mas em duas funções diferentes.

Por outro lado, até por serem uma abordagem muito recente, existem algumas limitações que devem ser tidas em conta ao optar pela utilização de uma arquitetura *serverless*. Embora algumas das limitações que se seguem possam, eventualmente, ser ultrapassadas, outras são inerentes aos conceitos da arquitetura.

- **Aplicações stateless**

  Cada vez que uma função recebe um pedido é invocada, é feito o processamento e depois deste estar concluído é novamente desligada. Isto significa que as aplicações não podem ter estado já que este não será partilhado nas próximas invocações. Veremos que este ponto terá um grande impacto na abordagem utilizada para gerir as notificações dos utilizadores, já será necessária a utilização de um *scheduler* para agendar as notificações sobre mudanças de estado dos dispositivos, o que não é possível numa aplicação *stateless*.

- **Funções de curta duração**

  O objetivo destas funções é serem de curta duração, caso contrário compensaria utilizar servidores *always-on* para alojar as aplicações. Aplicações que exijam elevados intervalos de tempo para processamento deverão ser evitadas nesta arquitetura.

Delineados os benefícios e limitações da utilização de uma arquitetura *serverless*, resta apenas deixar algumas considerações sobre a utilização da mesma para o caso de estudo da presente dissertação.

Tanto a aplicação conversacional como a aplicação smart home deverão tirar partido deste padrão arquitetural, sendo que serão compostas por funções autónomas, cada uma com o seu objetivo e independente das restantes. Relembremos que os pontos de entrada destas aplicações são intenções, também independentes e autónomas entre si, sendo que a invocação do processamento destas intenções corresponde a pedidos HTTP. Tirando o caso particular do agendamento de notificações para a mudança de estado dos dispositivos, que será abordado no capítulo seguinte, as restantes funções não precisam do armazenamento

de estado e, caso seja, o mesmo é feito através de uma base de dados existente na *Cloud Firestore*. Relembremos ainda que o *Dialogflow* armazena também informação sobre o estado atual da conversa com o utilizador, pelo que isto deixe de exigir esta necessidade em qualquer umas das aplicações anteriores.

As funções a implementar não serão responsáveis por fazer migrações ou outro tipo de operações que leve a elevados tempos de processamento, pelo que a limitação de serem obrigatoriamente de curta duração não terá qualquer impacto nas mesmas.

Por fim e sendo que um dos principais objetivos é conseguir uma arquitetura escalável, como vimos pelos pontos apresentados, através da utilização de uma arquitetura *serverless* estamos a garantir *autoscaling*, sendo que esta preocupação deixa de estar do lado do programador, sendo responsabilidade exclusiva do fornecedor do serviço.

7.3 COMMAND-QUERY RESPONSABILITY SEGREGATION

Para além da arquitetura *Serverless*, o *Command-Query Responsability Segregation* (CQRS) é outro design pattern cujo foco está na escalabilidade. Este padrão arquitetural sugere a utilização de diferentes modelos para realizar leituras e escritas. Antes de explicar em detalhe o que é CQRS, podemos começar por entender o que é o *Command-Query Separation* (CQS), já que este é a base para o CQRS.

7.3.1 *Command-Query Separation*

Este pattern foi introduzido por Bertrand Meyer em *Object Oriented Software Construction*[22] e evidencia que cada método deve ser um comando que executa uma ação sem retornar qualquer tipo de dados, ou uma query que, pelo contrário, só deverá retornar dados e não alterar o estado inicial, mas que nunca deverá ser ambos ao mesmo tempo.

- **Commandos** - alteram dados mas não os devolvem;
- **Queries** - devolvem dados mas não os alteram, logo nunca podem dar origem a *side-effects*.

Desta forma, deve existir uma separação clara entre métodos que alteram o estado dos que não o fazem, já que isto garante maior fiabilidade ao escrever aplicações, dado que as preocupações passam a ser concentradas nos comandos e é com estes que o estado é alterado.

No entanto nem tudo pode seguir este padrão, um exemplo muito simples seria a implementação de uma queue e/ou stack, já que ambas as implementações devolveriam um elemento ao mesmo tempo que alterariam o seu estado (*pop*).

7.3.2 *CQS vs CQRS*

Enquanto que o CQS funciona ao nível do método ou até classe, o CQRS aplica os conceitos base do CQS ao nível da aplicação[11]. De modo a garantir um sistema escalável e tendo em conta que com aplicações do género facilmente conseguimos chegar a milhões de dispositivos IoT a serem controlados em simultâneo, o CQRS pode facilitar isto com uma abordagem relativamente simples: separar as leituras e escritas em dois subsistemas distintos. Assim as leituras (queries) seriam realizadas em modelos diferentes dos das escritas (commands).

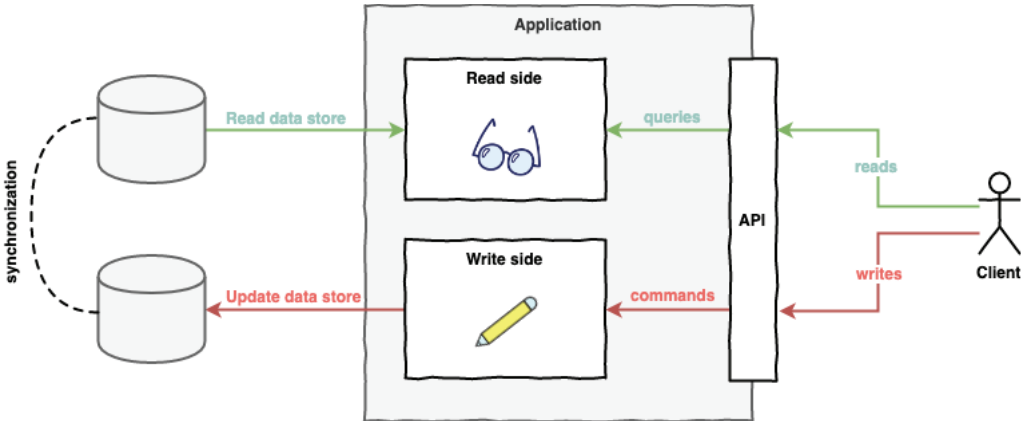


Figura 27: Demonstração da aplicação do design pattern na prática.

A Figura 27 demonstra este uso de dois subsistemas distintos em conjunto com duas bases de dados também diferentes entre si, sendo uma para os modelos de escrita e outra para os de leitura. Na figura podemos ver que as interações do utilizador se resumem a dois tipos diferentes, ou são consultas de informação (queries) ou alteração da informação existente (comandos), isto é, leituras ou escritas, respectivamente.

As leituras usam o subsistema *Read side* que lê a informação de uma *Read data store*, enquanto que as escritas usam o subsistema *Write side*, atualizando os dados presentes na *Update data store*. Para manter as bases de dados sincronizadas, já que as escritas têm de ser refletidas no subsistema de leitura, são usados eventos. Quando é feito uma atualização da informação existente na base de dados de escrita é emitido um evento que quando consumido pelo subsistema de leitura deverá atualizar a sua *data store*, garantindo que futuras leituras realizadas nesta retornem dados que reflitam as escritas que ocorreram até aquele momento.

Como os comandos não devolvem qualquer tipo de informação, devem existir mecanismos de validação dos mesmos para evitar, o mais cedo possível, possíveis erros que

possam acontecer durante o processamento do mesmo. Assim sendo, é recomendado que para cada comando criado, exista uma validação da informação contida no mesmo, algo que não acontece para as queries.

À primeira vista a aplicação de CQRS parece não trazer grandes benefícios, tirando a modularidade que ganhamos ao nível aplicacional, já que passamos a ter dois subsistemas que podem ser desenvolvidos de forma independente entre si. No entanto, a ideia do padrão é ter uma base simples que deve depois ser explorada em busca de arquiteturas robustas e escaláveis. Nesta secção apresentaremos os benefícios ganhos ao explorar algumas das alternativas existentes, como a utilização de duas bases de dados distintas para os dois modelos.

- **Subsistemas distintos**

  A segregação em dois subsistemas distintos permite uma divisão de responsabilidades mais coerente, sendo que um dos sistemas é responsável por atualizar a informação, alterando o estado do mesmo, enquanto que o outro apenas tem de fazer consultas a esse estado. Esta separação permite, por exemplo, desenvolver os sistemas em simultâneo, já que devem ser independentes entre si.

- **Base de dados distintas**

  A utilização de bases de dados distintas para os modelos de escrita e leitura é, provavelmente, o melhor benefício do qual podemos tirar partido com este padrão arquitetural. Com a utilização de duas bases de dados diferentes garantimos que as operações de escrita não impactam as de leitura e vice-versa. Isto por si já é uma grande vantagem, no entanto, é possível explorar um pouco mais este fator para não só usar duas bases de dados distintas mas também duas tecnologias diferentes para estas, uma situação que pode fazer sentido conforme os requisitos para as escritas e leituras de um determinado sistema, por exemplo se for necessário utilizar uma base de dados relacional para as escritas, enquanto que para as leituras seja necessária a utilização de uma base de dados NoSQL. Ou até podemos ter uma base de dados desenvolvida a pensar em leituras rápidas para o subsistema de leituras, como por exemplo ElasticSearch[6] onde os dados são indexados, garantindo uma excelente performance para leituras, em conjunto com uma tecnologia desenvolvida a pensar em escritas rápidas para o subsistema de escritas, como Cassandra[7] ou MongoDB[8], ficando com o melhor dos dois mundos.

- **Escalabilidade**

6 https://www.elastic.co/
7 http://cassandra.apache.org/
8 https://www.mongodb.com/

A separação em dois subsistemas permite que cada um destes seja escalável independentemente do outro, sendo que poderíamos até ter um número diferente de servidores para os diferentes sistemas. O desenvolvimento de cada um destes também pode ser independente, um fator extremamente relevante para tornar o sistema escalável. Por fim, e como vimos no ponto anterior, ao ter bases de dados distintas permitimos que cada uma destas escale de forma independente, podemos alocar mais máquinas para o subsistema de leitura se, por exemplo, existir um elevado número de leituras relativamente ao número de escritas.

Como seria de esperar, com a aplicação do padrão arquitetural nas aplicações, para além de benefícios devemos ainda levantar algumas considerações sobre as limitações e/ou desvantagens que surgem.

- **Esforço adicional**

  Com a introdução de dois subsistemas em vez de um único estamos a introduzir também complexidade na arquitetura da aplicação e consequentemente será necessário um esforço adicional por parte dos developers, não só na adaptação à utilização do design pattern mas também durante a sua implementação.

- **Consistência eventual**

  A principal limitação com a introdução deste padrão, em particular com a utilização de duas bases de dados distintas para os sistemas de escrita e leitura é a necessidade de sincronização entre estas. Esta sincronização faz com que o sistema passe de *Strong Consistency* para *Eventual Consistency*[11]. Ao termos um sistema eventualmente consistente garantimos que, se não fizemos mais nenhuma alteração num determinado conjunto de dados, inevitavelmente todas as leituras a este conjunto de dados irão devolver o valor da última alteração realizada.

  Se considerarmos a Figura 28 como exemplo percebemos facilmente como estamos a perder consistência nos dados do sistema. O cliente começa por atualizar o valor de X para 1, cujo estado inicial é 0 em ambas as bases de dados, tal como é refletido nos pontos **1** e **2**. Antes da sincronização entre as bases de dados de escrita e leitura estar finalizada, o mesmo ou até um outro cliente realiza uma operação de leitura ao valor de X (ponto **3**) que, como podemos observar pelo ponto **4** devolve o valor inicial de X. A sincronização entre as bases de dados é a última operação a ser realizada (ponto **5**), pelo que todas as futuras leituras ao valor de X irão refletir a sua última alteração.

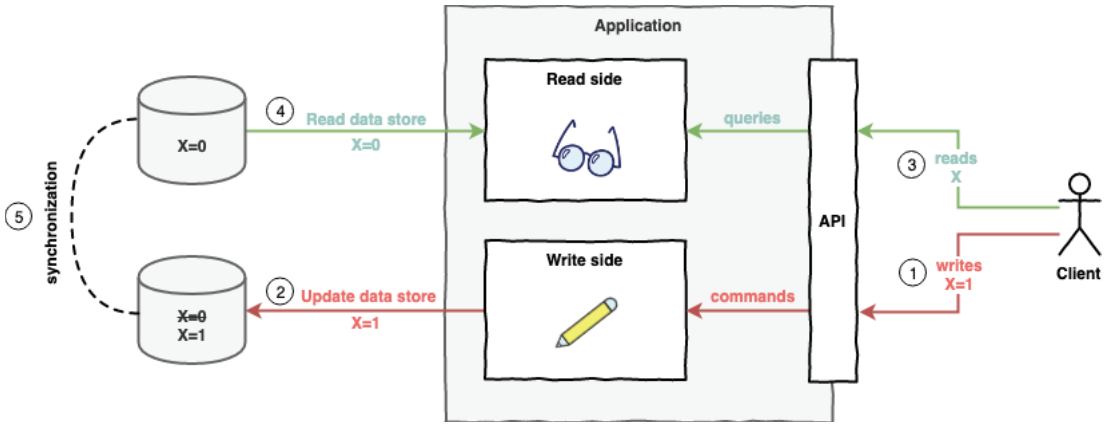


Figura 28: Exemplo da perda de consistência no sistema.

> Em vez de assumir que o sistema está sempre num estado consistente, aceitamos que este o alcançará num determinado ponto no futuro. Não importa se demora 1 ms ou até 1 ano, ao ter um sistema *eventually consistent* garantimos que este sistema chegará a um ponto consistente no futuro, sem nunca especificar quanto tempo levará até que este ponto chegue efectivamente.

A utilização deste padrão arquitetural em aplicações para assistentes digitais cujo o objetivo seja a gestão de dispositivos inteligentes revela-se útil não só por garantir escalabilidade a estas aplicações, mas também porque as suas limitações não trazem compromissos muito significativos para estas aplicações.

Se considerarmos como exemplo o facto da utilização de comandos não devolver resultados, o que significa que, por exemplo, se uma operação para mudar o estado de um determinado dispositivo falhe, isto não poderá ser comunicado imediatamente ao utilizador. No entanto, isto já seria o comportamento esperado ao utilizar estas aplicações. Se um utilizar tentar desligar a sua televisão inteligente com *Turn my TV off*, deverá esperar uma resposta imediata por parte do assistente, já que não fará qualquer sentido deixar o utilizador à espera que a operação seja realmente realizada. No caso de falha da operação é esperado que o assistente já tenha respondido ao utilizador com *Ok, I'm turning your TV off*, e portanto, o mais correcto será enviar uma notificação para o mesmo mencionando que entretanto ocorreu uma falha a realizar a sua operação. Assim sendo, a inexistência de retorno aquando uma operação que resulte num comando, isto é, uma operação de escrita, é mais importante devolver imediatamente uma resposta ao utilizador, independentemente da operação ter sido realizada com sucesso ou não, do que o fazer esperar, causando uma experiência de conversação pouco realista com o assistente.

É de notar que, para este exemplo, estamos a considerar uma validação prévia dos comandos, isto é, se um comando for construído com informação inválida, isto é detetado antes de começar o processamento do mesmo, pelo que o envio de notificações só deverá acontecer quando para um comando válido é impossível satisfazer o pedido inicial do utilizador (e.g. falha na comunicação entre a aplicação e o dispositivo).

7.4 Arquitetura da aplicação smart home

Com os padrões arquiteturais mais relevantes já definidos, iremos agora partir para a definição da arquitetura da aplicação de smart home. É de notar que esta será a aplicação que contém não só a API descrita em 6.5, mas também a própria API exigida pela Google para controlo das intenções de smart home, previamente apresentadas em 4.8.1. Além disto, esta aplicação não tem qualquer contacto com o módulo de NLP, o que levará a uma arquitetura com um foco diferente da arquitetura da aplicação conversacional.

A arquitetura apresentada a seguir é muito semelhante à arquitetura em camadas geralmente utilizada em *Domain-driven design*[10], também conhecido como DDD. Embora o conceito de DDD tenha maior relevância no contexto de *Object-oriented programming*, mais recentemente é possível ver este a ser aplicado também para a construção de arquiteturas focadas em *microservices*[29], onde as linguagens multi-paradigma (como é o caso do JavaScript) têm cada vez mais impacto. Apesar desta metodologia de desenvolvimento não fazer premissas ao nível da construação da arquiteturas das aplicações, existe uma arquitetura em camadas característica das aplicações que seguem esta metodologia de desenvolvimento[24]. Deste modo, começaremos por explicar cada uma das camadas que constituem a aplicação, identificando para cada uma destas os seus componentes.

Na Figura 29 é apresentada a arquitetura da aplicação smart home. O design pattern CQRS deverá estar refletido nesta arquitetura, havendo uma separação lógica entre os sistemas de escrita e leitura. Esta separação, como veremos adiante, estará representada nos diagramas com duas cores distintas, sendo o verde para as leituras/queries e o vermelho para as escritas/comandos e afectará diferentes camadas da aplicação.

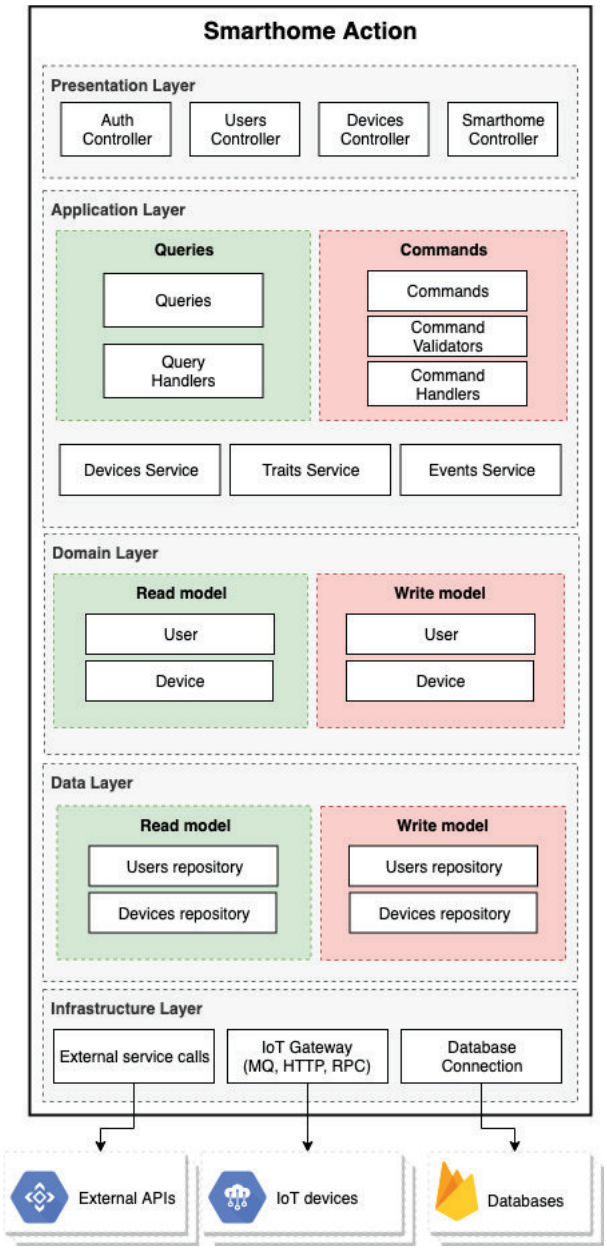


Figura 29: Arquitetura da *Smart home Action*.

7.4.1 *Presentation layer*

A *Presentation layer* é o ponto de entrada na aplicação, nesta camada temos os componentes que refletem precisamente isso, tais como controladores de APIs REST ou até consumidores de um serviço de mensagens. No caso da arquitetura apresentada temos os seguintes controladores:

- **Auth controller**

  Controlador responsável por ter lógica associada à autenticação dos utilizadores. As aplicações de smart home exigem um mecanismo de autenticação dos utilizadores para garantir que cada utilizador controla apenas os seus dispositivos.

- **Users controller**

  Neste controlador estará a lógica associada às rotas dos utilizadores definidas na especificação da API em 6.5.

- **Devices controller**

  Neste controlador estará a lógica associada às rotas dos dispositivos definidas na especificação da API em 6.5.

- **Smart home controller**

  O Smart home controller é responsável por receber os pedidos relativos a intenções smart home (Execute, Query, Sync e Disconnect). Se a API para gestão de dispositivos, cujo objetivo é adicionar funcionalidade ao assistente, não tivesse sido adicionada, este controlador e o de autenticação seriam os únicos necessários para fazer a gestão do estado dos dispositivos.

A única lógica presente nesta camada será a criação de comandos ou queries para fazer o processamento dos mesmos, sendo que este processamento é feito na *Application layer*. A criação de comandos e queries deve respeitar os design patterns anteriormente apresentados, na medida em que um pedido GET será traduzido numa query, enquanto que um pedido POST, PUT ou DELETE num comando.

7.4.2 *Application layer*

A *Application layer* é onde a separação lógica entre comandos e queries é introduzida na aplicação. Nesta camada devem estar presentes os comandos e queries disponíveis para utilização pela *Presentation layer*, assim como os componentes responsáveis por fazer o processamento destes, denominados de *handlers*. Se considerarmos a existência de um comando *CreateUserCommand*, deverá existir um *CreateUserCommandHandler* onde a lógica de execução do comando estará presente. Os *handlers* deverão fazer uso tantos dos serviços presentes nesta camada, como das camadas seguintes (*Domain* e *Data*) para, seguindo o mesmo exemplo, utilizar o repositório de utilizadores para adicionar efetivamente o utilizador criado à base de dados.

Dado que um comando não devolve qualquer tipo de informação, é importante que no momento de criação destes exista algum tipo de validação para verificar que durante o

seu processamento não irão existir erros, já que ao nada ser retornado é impossível saber se a operação foi bem sucedida ou não. Tendo em conta isto, cada comando deverá ter uma validação, para o exemplo anterior deveria existir um *CreateUserCommandValidator* que fazia a verificação durante a sua criação para evitar à cabeça possíveis erros durante o seu processamento.

Nesta camada podemos ainda encontrar três serviços, com responsabilidades distintas:

- **Devices service**

  Este serviço só deverá ser usado por comandos e queries que atuem sobre o estado dos dispositivos, já que a sua responsabilidade é reconhecer o tipo de dispositivo (segundo os tipos de dispositivos vistos em 4.8.2) para o qual a operação se trata e a partir deste utilizar o *Traits service* para processar as operações para a funcionalidade (*trait*) correspondente.

- **Traits service**

  Este serviço é responsável por reconhecer a funcionalidade correspondente da operação (segundo os tipos de funcionalidades vistos em 4.8.3) e efetuar a lógica necessária para uma determinada funcionalidade.

- **Events service**

  Vimos anteriormente que seria necessário algum tipo de mecanismo de sincronização entre os dois subsistemas. Este mecanismo de sincronização será através de eventos, isto é, quando um utilizador é criado ou o estado de um dispositivo é alterado, a escrita é refletida no modelo de escrita e na base de dados correspondente. Posteriormente a este acontecimento é utilizado o *Events service* que irá emitir um evento com a informação sobre a operação realizada, por exemplo *UserCreatedEvent* ou *DeviceStateChangedEvent*, este evento deverá ser consumido pelo modelo de leitura que por sua vez irá atualizar a sua base de dados para ficar com a informação consistente à do modelo de escrita.

7.4.3 *Domain layer*

A *Domain layer* é onde estão presentes as entidades do domínio. As entidades devem estar replicadas para cada os modelos de escrita e leitura, tal como podemos observar no diagrama. Para efeitos de simplicidade as únicas entidades presentes serão o utilizador e o dispositivo, sendo que estas podiam ser exploradas de modo a criar entidades mais pequenas e autónomas. Validações sobre estas entidades devem estar presentes nesta camada.

7.4.4 *Data layer*

Do mesmo modo que na *Domain layer*, a *Data layer* deve apresentar uma clara divisão entre os modelos de escrita e leitura. As entidades do modelo de escrita da camada anterior deverão ser usadas nos repositórios do modelo de escrita desta camada, sendo que o mesmo se aplica para os modelos de leitura em ambas as camadas.

Ainda como na camada anterior, os repositórios deverão ser duplicados para os diferentes modelos. Note-se que, no entanto, cada repositório deverá ter operações de uma determinada entidade para uma base de dados específica, isto é, os repositórios do modelo de leitura e do modelo de escrita vão utilizar tecnologias de bases de dados diferentes, o que implica que a lógica contida nos mesmos não seja na realidade duplicada entre os modelos.

A criação dos repositórios, tanto para os utilizadores como dispositivos, é baseada no *Repository Pattern*[12] cujo o objetivo é a abstração de detalhes de implementação, como o tipo de bases de dados a ser utilizada. Isto significa que, apesar dos repositórios terem implementações distintas para os modelos de leitura e escrita, deverão ser utilizados pelas camadas superiores segundo a mesma interface.

7.4.5 *Infrastructure layer*

Por fim, a *Infrastructure layer* terá configurações partilhadas por todas as restantes camadas. Além disso pedidos a APIs externas, gateways para os dispositivos IoT e conexões a base de dados serão responsabilidade desta que é a camada mais *low-level* da aplicação.

7.5 Arquitetura da aplicação conversacional

A arquitetura da ação de conversação (ou *Conversational Action*) é relativamente diferente da anterior. O objetivo desta aplicação não é a gestão de milhares ou até milhões de dispositivos inteligentes como a anterior, mas sim servir de *middleware* para permitir a utilização de invocações customizadas e tirar partido de funcionalidades acrescidas para o assistente. O facto desta aplicação ter um módulo de NLP para comunicação com o Dialogflow faz com que a sua arquitetura tenha particularidades distintas das da *Smart home Action*, levando a uma abordagem diferente ao nível arquitetural.

O propósito desta aplicação, tal como o nome da própria indica, é a conversação com o utilizador. Desta forma, a lógica presente nesta é precisamente no contexto de criar uma conversa com o utilizador e tirar partido de diferentes fluxos de conversação para garantir que este consegue realizar as ações que pretende. A lógica para gestão de informação e

gestão dos dispositivos não é sua responsabilidade, pelo que, o correto será usar a *Smart home Action* para o efeito.

Para esta aplicação, ao nível arquitetural, não serão especificadas as camadas e o foco será direcionado para os componentes que devem estar, obrigatoriamente, presentes para o seu correto funcionamento. A arquitetura proposta deve ser modular, existindo uma segregação clara de responsabilidades para cada um dos componentes que a compõem, mas também deverá ser pensada de modo a ser independente da tecnologia utilizada, (salvo exceções em alguns dos componentes, que como veremos mais adiante serão sempre dependentes da tecnologia do assistente), permitindo que a sua portabilidade para outros assistentes não seja comprometida.

Na Figura 30 é apresentada a arquitetura proposta para as aplicações de conversação, aliadas aos restantes componentes do sistema com que mantêm contacto, por exemplo a *Smart home Action*.

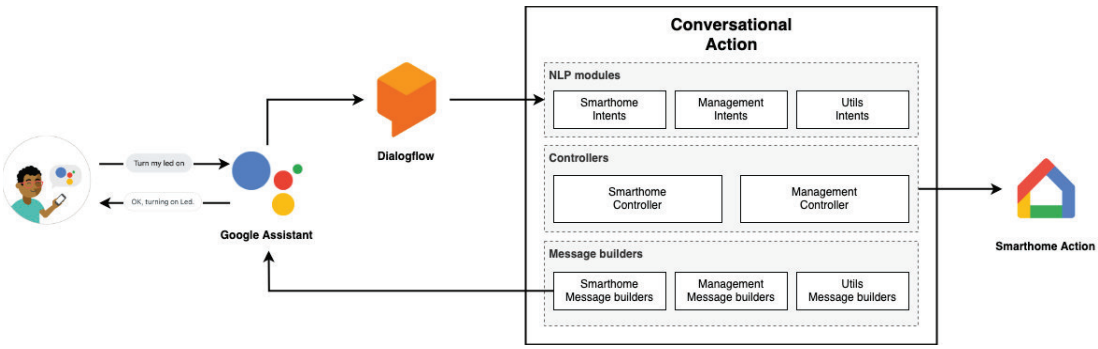


Figura 30: Arquitetura da *Conversational Action*.

7.5.1 *Módulos NLP*

Os módulos de NLP serão os módulos de *Natural Language Processing* que vão ter contacto direto com o Dialogflow. Quando o utilizador envia uma mensagem para a aplicação, esta é primeiramente processada pelo Dialogflow que, de acordo com os modelos treinados, irá retirar informação da mensagem do utilizador - intenções e entidades - que são comunicadas à aplicação conversacional. Os módulos de NLP são os que têm esta comunicação com a plataforma Dialogflow, sendo que existirá uma divisão de responsabilidades, conforme o tipo de intenções, para cada módulo.

- **Intenções de Smart home**: intenções para consulta e controlo do estado dos dispositivos;
- **Intenções de Gestão**: também chamadas de *Management intents* são intenções para gestão de informação sobre os dispositivos;

- **Intenções utilitárias**: também chamadas de *Utils intents* são intenções auxiliares para saudação, pedidos de ajuda, subscrição de notificações, etc.

7.5.2 *Controladores*

Os controladores ou *controllers* contêm lógica associada ao processamento de invocações relacionadas com o controlo do estado dos dispositivos (*Smart home Controller*) ou gestão de informação sobre estes (*Management Controller*). Estas funcionalidades exigem comunicação com a Smart home Action, pelo que nestes componentes devem ser realizados os pedidos a esta aplicação, fazendo o processamento das respostas resultantes. De um modo geral, a utilização de APIs externas e a lógica associada a esta devem estar presentes nestes componentes.

7.5.3 *Message builders*

Assim que os pedidos a APIs externas são realizados e as suas respostas processadas é necessário criar uma resposta para os utilizadores, é aqui que entram os *Message Builders*. O único propósito destes será, conforme os resultados obtidos nos controladores, construir respostas para os utilizadores. A construção de mensagens por parte dos *Message Builders* deverá fazer uso das funcionalidades do assistente digital que está a ser utilizado e, portanto, ser dependente da tecnologia utilizada (e.g. a construção de uma lista de itens para apresentação no ecrã do utilizador é diferente entre o Google Assistant e a Alexa). Se no futuro fosse necessário tornar a aplicação disponível para outros assistentes digitais que não o Google Assistant, teriam de ser adicionados *Message Builders* tirando partido da tecnologia dos novos assistentes.

8

IMPLEMENTAÇÃO

Evidenciada a arquitetura do sistema, segue-se a descrição dos aspectos da implementação mais relevantes. Neste capítulo serão apresentadas com mais detalhe as tecnologias necessárias e abordados os desafios que surgiram durante a implementação do sistema.

8.1 TECNOLOGIAS

De seguida serão apresentadas as tecnologias utilizadas durante a implementação das aplicações. Veremos que embora a linguagem utilizada (JavaScript) e o *runtime environment* (Node.js) seja o mesmo para todas as aplicações, as frameworks das quais cada uma destas tirou partido são diferentes e têm objetivos muito distintos entre si.

Os motivos que levaram à escolha de cada uma das tecnologias escolhidas deverá estar presente nesta secção, assim como uma breve apresentação da utilização de cada uma destas. Explicações detalhadas e extensas deverão ser consultadas nas documentações oferecidas pelas próprias, já que a sua explicação não traria qualquer valor para o contexto da dissertação. É de notar que os principais objetivos a alcançar são a possibilidade de uma implementação rápida, mas que oferece garantias de extensibilidade para trabalho futuro e ainda, tal como foi para as arquiteturas das aplicações, a possibilidade de escalar as aplicações implementadas.

8.1.1 *JavaScript e Node.js*

A opção por JavaScript, em particular Node.js, foi devida à proximidade desta com a documentação fornecida pela própria Google para o desenvolvimento de aplicações tanto de conversação como de smart home. Embora não tenha tido qualquer experiência com a linguagem até ao desenvolvimento destas aplicações, das linguagens disponíveis pelos SDKs da Google para o desenvolvimento destas aplicações, era a única compatível com o desenvolvimento de Cloud Functions, na sua infraestrutura - Google Cloud.

Node permite um rápido desenvolvimento de APIs REST através de frameworks como o Express[1], o que se torna ideal para o desenvolvimento das aplicações protótipo. Para além disto, Node tem a particularidade de ser *single-threaded* sendo que cada pedido é processado pela mesma thread, exigindo que o processamento durante a execução de cada um destes seja, por padrão, assíncrono. Esta propriedade permite que aplicações desenvolvidas em Node facilmente suportem milhares de conexões em simultâneo, mesmo utilizando uma única thread para o efeito. Na secção 8.1.2 falaremos da importância que esta propriedade tem para a construção de aplicações escaláveis.

Por outro lado, reconhecemos já em capítulos anteriores que as aplicações não vão realizar qualquer tipo de operações cujo tempo de processamento e utilização de CPU seja elevado. Para estas situações a escolha de Node não se justificaria e a performance das aplicações iria sofrer grandes penalizações. É importante retirar que a escolha por este *runtime environment* para as aplicações a desenvolver é devida à escalabilidade que estas devem apresentar em conjunto com elevado número de operações de IO que vão realizar, onde a programação assíncrona terá um grande impacto.

8.1.2 *Programação assíncrona*

Através da utilização de programação assíncrona para operações de IO permitimos que estas operações não estejam a bloquear a thread e esta possa servir outros pedidos, não ficando assim a bloqueá-los como acontece com a utilização de operações síncronas. A utilização de programação assíncrona para o desenvolvimento de aplicações escaláveis é certamente uma mais valia.

Isto é relevante já que todo o código aplicacional desenvolvido fará uso da programação assíncrona. O desenvolvimento de aplicações para o contexto smart home exige a utilização de operações de IO, seja para fazer pedidos HTTP entre as aplicações, chamadas a base de dados ou até quando é utilizada comunicação através de mensagens.

8.1.3 *Google JavaScript Style Guide*

De modo a garantir consistência no código desenvolvido foram utilizadas as regras e *standards* definidos como ideias pela Google para o desenvolvimento em JavaScript. Estes *coding standards* podem ser encontrados no próprio *Google JavaScript Style Guide*[2], que têm disponível para consulta e para utilização por parte de qualquer programador de JavaScript.

1 https://expressjs.com/
2 https://google.github.io/styleguide/jsguide.html

Para garantir que estes standards eram aplicados no código de cada uma das aplicações foi utilizada a ferramenta ESLint[3].

8.1.4 *Cloud Functions*

Já falamos do porquê da utilização desta ferramenta e quais as vantagens e desvantagens que surgem com a utilização da mesma em 7.1.3, pelo que, nesta secção será realizada uma pequena demonstração de como se tira partido desta ferramenta para construir efetivamente *Cloud Functions*, embora isto esteja detalhamente explicado na própria documentação[4].

No da Figura 31 é demonstrado como se expõem a API de uma aplicação, utilizando a framework Express[5] através de uma *Cloud Function*. A API criada disponibiliza apenas uma rota para pedidos GET, pelo que, a mesma deverá estar disponível através da rota /api/ da aplicação.

```
const functions = require('firebase-functions');
const express = require('express');

const app = express();

app.get('/', (_, response) => {
    response.send('Smarthome conversational action APIs.');
});

exports.api = functions.https.onRequest(app);
```

Figura 31: Exemplo da criação de uma Cloud Function.

8.1.5 *Firebase*

Assumindo a existência de uma aplicação funcional, seria alguma possível fazer deploy da mesma para a infraestrutura da Google. Para realizar este deploy foi utilizado o Firebase CLI[6] que permite a utilização de diversos comandos para gestão das aplicações através de uma *Command-line interface*. Depois de realizado o deploy da aplicação é possível consultar múltiplas dashboards na plataforma Firebase com informações relativas ao estado das *Cloud Functions* (Figura 32), dados em real-time presentes na *Cloud Firestore* (Figura 33), estatísticas, monitorização, etc.

3 https://github.com/eslint/eslint
4 https://firebase.google.com/docs/functions/http-events
5 https://expressjs.com/
6 https://firebase.google.com/docs/cli

Figura 32: Dashboard das Cloud Functions.

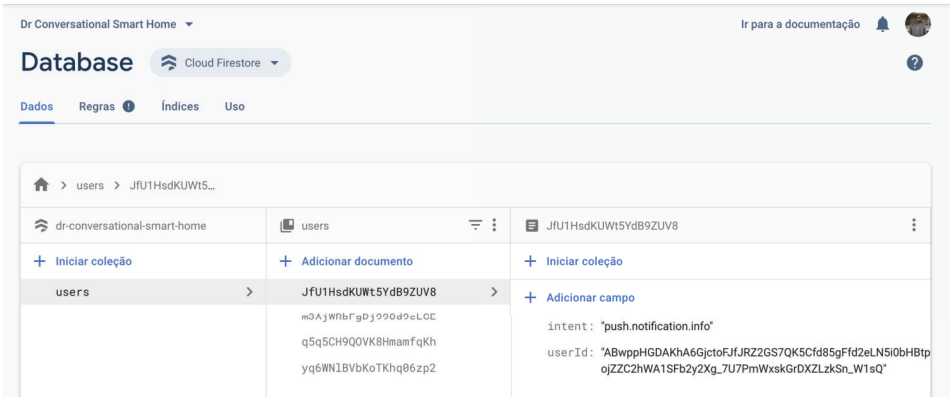


Figura 33: Dashboard da Cloud Firestore.

8.2 Desenvolvimento das aplicações

Nesta secção será explicada a implementação de cada uma das aplicações para garantir um sistema funcional e capaz de satisfazer os requisitos inicialmente definidos pela arquitetura que se está a propor. Para cada uma das aplicações que se seguem será feita referência aos desafios e problemas encontrados, em conjunto com as soluções implementadas para resolução destes. A ordem pela qual as aplicações serão explicadas transparece a ordem pela qual as mesmas foram desenvolvidas, de modo a facilitar a compreensão da sua implementação. Note-se, no entanto, que muitos detalhes de implementação serão abstraídos já que o objetivo será dar uma visão geral das decisões que foram tomadas no momento de implementação e não todos os detalhes desta.

8.2.1 *Smart Home Action*

Antes de serem desenvolvidas funcionalidades acrescidas nesta aplicação é necessário ter uma aplicação base que permita aos utilizadores fazer a gestão do estado dos seus dis-

positivos inteligentes. Esta aplicação base funcionará como pré-requisito para as diversas funcionalidades que de seguida serão implementadas sob a mesma.

A implementação desta aplicação base deverá seguir os padrões recomendados pela Google em questões de autenticação, processamento das diferentes intenções e até a correta utilização da *Home Graph*. Estes conceitos foram já explicados em 4.8, pelo que o objetivo neste capítulo será a implementação dos mesmos.

Configurações na plataforma Actions on Google

A plataforma *Actions on Google* permite a gestão de todo o tipo de ações, sejam estas de conversação ou de smart home. É através desta plataforma que podemos criar ações e escolher a configuração que será usada. O primeiro passo será precisamente este, na plataforma deveremos criar uma *smart home action* indicando o nome que será utilizado.

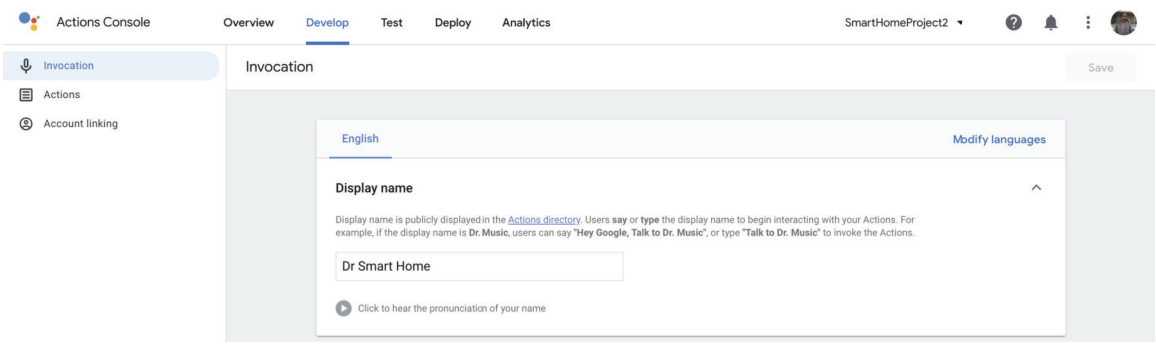


Figura 34: Definição do nome da ação na plataforma *Actions on Google*.

Posteriormente deverá ser indicado o URL do *fulfillment* da nossa aplicação. A rota ou *endpoint* introduzido deverá corresponder ao da *Cloud Function* responsável por fazer o processamento das intenções de smart home (Sync, Query, Execute, Disconnect), neste caso estará disponível através da rota /smarthome. É de notar que o *hostname* usado no exemplo da Figura 35 é apenas um exemplo e deveria ser substituído pelo *hostname* real da aplicação.

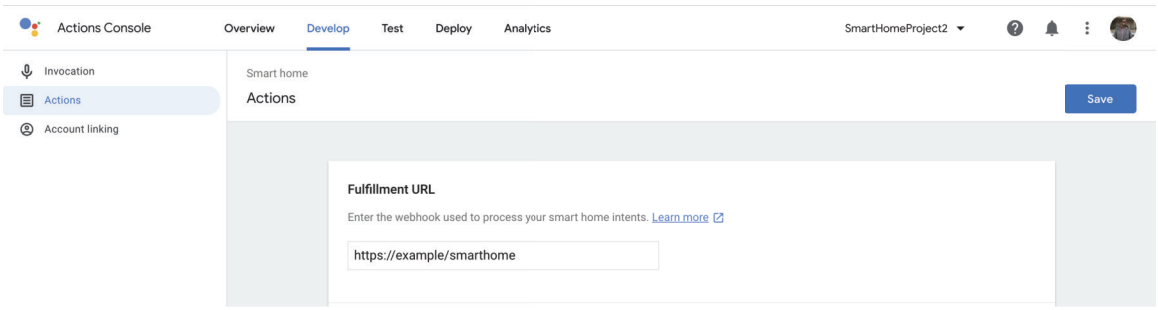


Figura 35: Configuração do URL do *fulfillment* da ação na plataforma *Actions on Google*.

Por fim é necessário fazer a configuração do *Account Linking*, isto será responsável pela autenticação dos utilizadores na aplicação já que só assim serão capaz de controlar o estado dos seus dispositivos. A autenticação utilizada não poderia ser usada num contexto real já que está a utilizar o mesmo código de autorização (*authorization code*) para qualquer tentativa de autenticação no sistema. Pela Figura 36 podemos verificar que são especificadas duas rotas, correspondentes a duas *Cloud Functions* distintas, estas não serão explicadas com grande detalhe já que estão simplesmente a simular uma autenticação utilizando *tokens* falsos.

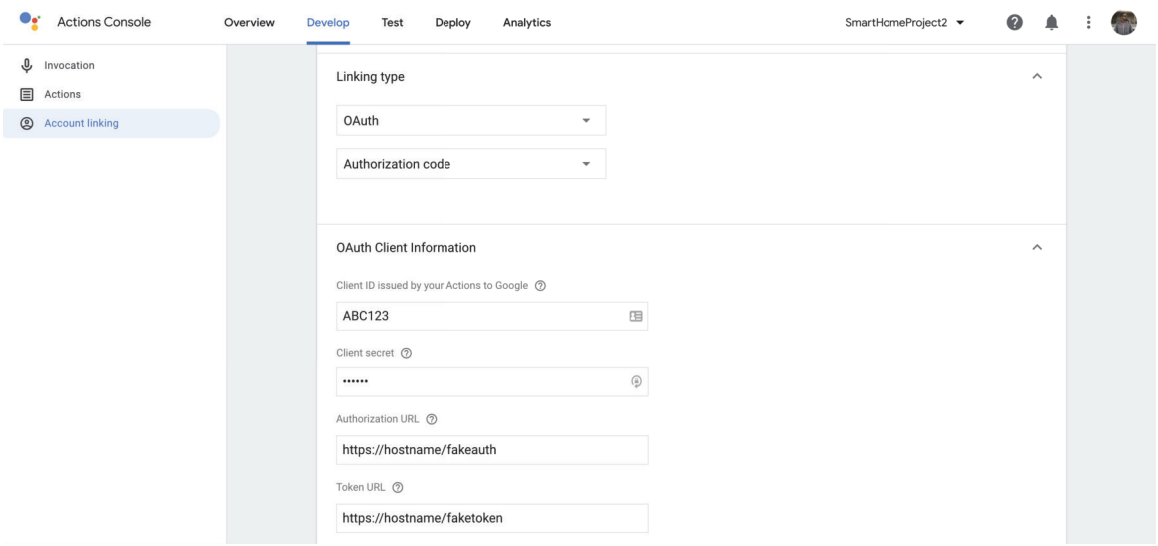


Figura 36: Configuração do *Account Linking* da ação na plataforma *Actions on Google*.

Intenções Smart Home

Com a configuração inicial realizada e a autenticação a funcionar devidamente, segue-se a implementação das intenções smart home que constituem a base das interações com

este tipo de aplicações. A implementação das intenções deverá seguir com os padrões arquiteturais previamente definidos.

Para fazer o processamento destas intenções, teremos um controlador *SmarthomeController* que fará, para cada uma das intenções, a criação da respetiva query ou comando e o processamento dos mesmos, tal como é possível ver na Figura 37.

```
SmarthomeController.app = smarthome({
    debug: true,
    key: SMARTHOME_KEY,
    jwt: require('../../../infrastructure/configs/key.json'),
});

SmarthomeController.app.onSync( async (body) => {
    const query = GetDevicesQuery.build(userId);
    const devicesPayload = await GetDevicesQueryHandler.handle(query);

    return {
        requestId: body.requestId,
        payload: {
            agentUserId: '123',
            devices: devicesPayload,
        },
    };
});

SmarthomeController.app.onQuery( async (body) => {
    const query = GetDeviceStateQuery.build(body);
    const response = await GetDeviceStateQueryHandler.handle(query);

    return response;
});

SmarthomeController.app.onExecute( async (body) => {
    const command = UpdateDeviceStateCommand.build(body);
    const response = await UpdateDeviceStateCommandHandler.handle(command);

    return response;
});
```

Figura 37: Excerto do código-fonte do *SmarthomeController*.

1. **SYNC**

   A intenção SYNC deverá resultar numa resposta, em formato JSON, com as informações dos dispositivos do utilizador. Nestas informações deve estar contido o tipo de dispositivo e as funcionalidades que dispõe. Para fazer o processamento desta intenção deverá ser utilizado o *handler onSync*. Notemos que para efetuar o processamento do pedido recebido e como o objetivo será consultar as informações de todos os dispositivos para um determinado utilizador, é criada a query *GetDevicesQuery* e feito o processamento da mesma através do seu *handler*.

   O processamento desta query será ir à base de dados do subsistema de leitura consultar os dispositivos do utilizador. Especial atenção para a utilização do *DevicesReadMo-*

delRepository que deverá, na sua implementação, utilizar uma base de dados diferente da do modelo de escrita.

```
GetDevicesQueryHandler.handle = async (query) => {
    return await DevicesReadModelRepository.getDevices(query.user);
};
```

Figura 38: Excerto do código-fonte do *GetDevicesQueryHandler*.

2. **QUERY**

A intenção de QUERY reflete a vontade do utilizador de consultar o estado de um dos seus dispositivos. Do mesmo modo que na intenção anterior este pedido deve ser refletido numa query. Neste caso a query será *GetDeviceStateQuery* e o seu *handler* irá utilizar o *DevicesService* para, conforme o tipo de dispositivo, consultar o seu estado.

```
GetDeviceStateQueryHandler.handle = async (getDeviceStateQuery) => {
    const { requestId, body } = getDeviceStateQuery;
    const queryPromises = [];
    const payload = {
        devices: {},
    };

    for (const input of body.inputs) {
        for (const device of input.payload.devices) {
            let queryDevice;
            queryDevice = DevicesService.executeQuery(device.id, payload);
            queryPromises.push(queryDevice);
        }
    }

    return Promise.all(queryPromises).then((_) => (
        {
            requestId: requestId,
            payload: payload,
        })
    );
};
```

Figura 39: Excerto do código-fonte do *GetDeviceStateQueryHandler*.

3. **EXECUTE**

A intenção de EXECUTE será recebida quando o utilizador pretender alterar o estado de um determinado dispositivo. Ao contrário das restantes intenções, esta não será uma funcionalidade para consulta de informação mas sim para alteração desta, o que implica a utilização de um comando em vez de uma query. O comando a utilizar será o *UpdateDeviceStateCommand*. O *handler* deste comando, apresentado na Figura 40, deverá utilizar o *DevicesService* que fará uso do *TraitsService* para efetuar o processamento para o trait do dispositivo que o utilizador pretende alterar o estado. Se considerarmos como exemplo um utilizador que pretende mudar o estado de um led

de ligado (ON) para desligado (OFF), então será utilizado o componente *OnOffService*, que terá a lógica para efetivamente desligar o led.

```
UpdateDeviceStateCommandHandler.handle = async (updateDeviceStateCommand) => {
    const { requestId, body } = updateDeviceStateCommand;

    const payload = {
        commands: [{
            ids: [],
            status: 'SUCCESS',
            states: {
                online: true,
            },
        }],
    };
    const statesPayload = {};

    for (const input of body.inputs) {
        for (const command of input.payload.commands) {
            for (const device of command.devices) {
                payload.commands[0].ids.push(device.id);
                const deviceState = await DevicesService.executeCommand(device.id, payload, command);
                statesPayload[device.id] = deviceState;
            }
        }
    }

    return {
        requestId: requestId,
        payload: payload,
    };
};
```

Figura 40: Excerto do código-fonte do *UpdateDeviceStateCommandHandler*.

Na Figura 41 é demonstrada a lógica que neste caso o *OnOffService* terá. O primeiro passo será atualizar o estado do dispositivo utilizando o repositório do subsistema de escritas. Posteriormente é utilizada uma *gateway* para atualizar o estado do dispositivo físico, ou seja, é enviado um pedido à aplicação do Raspberry Pi para que esta mude o estado dos seus periféricos. Por fim, é emitido um evento, utilizando o *DeviceStateChangedEmitter* para que o subsistema de leitura reflita a alteração que foi feita no de escrita, de modo a ficarem consistentes entre si.

```
OnOffService.handleCommand = async (deviceId, payload, command) => {
    payload.commands[0].states.on = command;

    // Update the device state in the write model repository
    await DevicesStateWriteModelRepository
        .updateDeviceState(deviceId, OnOffService.TRAIT, OnOffService.COMMAND, command);

    // Use the device gateway to change the physical device state
    await DeviceGateway.send(deviceId, command);

    // Emit an event to update the read model repository
    DeviceStateChangedEmitter.emit('device-state-changed', {
        deviceId: deviceId,
        traitName: OnOffService.TRAIT,
        commandName: OnOffService.COMMAND,
        params: command,
    });
};
```

Figura 41: Excerto do código-fonte do *OnOffService*.

Desenvolvimento da API

Depois de implementada toda a lógica necessária para fazer o processamento de cada uma das *smart home intents* temos a aplicação base desenvolvida e esta já pode ser utilizada pelos utilizadores, embora apenas com as frases de invocação resultantes do módulo NLP treinado pela Google. O próximo passo necessário será fazer a implementação da API, previamente especificada em 6.5.

Enquanto que através das intenções de *smart home* é possível gerir o estado dos dispositivos, a API permite gerir a informação sobre cada um destes e dos seus utilizadores, permitindo, por exemplo, a introdução de novas funcionalidades para um determinado dispositivo. Esta API permitirá também adicionarmos algumas funcionalidades à aplicação, como será o caso da consulta das funcionalidades que cada dispositivo dispõe que, de outra forma, não seria possível. Na implementação da API teremos dois controladores:

- **Controlador dos utilizadores**: responsável pelas operações CRUD dos utilizadores;
- **Controlador dos dispositivos**: responsável pelas operações CRUD dos dispositivos;

Para explicar detalhadamente o processamento realizado para cada uma das operações disponibilizadas pela API, seguiremos o exemplo da criação de um utilizador. No controlador dos utilizadores deve estar disponível a definição de um *handler* para métodos POST, já que este será o tipo de pedido utilizado durante a criação de novos utilizadores, representado na Figura 42. Neste *handler* deve ser criado o comando *CreateUserCommand* e chamado o respetivo *handler* (*CreateUserCommandHandler*) para fazer o processamento do mesmo.

```
router.post('/:id', async (request, response) => {
    const id = request.params.id;
    const user = request.body;

    const command = CreateUserCommand.build(id, user);
    await CreateUserCommandHandler.handle(command).catch((err) => {
        response.status(400).send(err.message);
        console.log(err);
    });
    response.status(201).send(`The user with the id ${id} was created.`);
});
```

Figura 42: Código-fonte do handler de pedidos POST no *UsersController*.

O *CreateUserCommandHandler*, na Figura 43 deverá receber o comando previamente criado e validar o mesmo, já que se este for construído com informação inválida deverá retornar imediatamente um erro impedindo o resto do processamento. Na criação de um utilizador deverá assim ser realizado algum tipo de validação para verificar que o *id* não está já associado a outro utilizador. Dadas as validações, é possível criar o modelo *User* com as informações contidas no comando recebido e inserir este na base de dados, utilizando o *UsersWriteModelRepository*, isto é, o subsistema de escrita.

Por fim, deverá ser emitido um evento *UserCreated* com as informações presentes no comando para que o subsistema de leitura garanta a consistência entre ambas as bases de dados.

```
CreateUserCommandHandler.handle = async (command) => {
    const { error } = UserCommandsValidator.validate(command);
    if (error) throw new Error(error.details[0].message);

    const checkUser = await UsersWriteModelRepository.getUser(command.id);
    if (checkUser) throw new Error(`User with id ${command.id} already exists.`);

    const user = new User(
        command.id,
        command.user.name,
        command.user.email,
        command.user.access_token
    );

    await UsersWriteModelRepository.insertUser(user);

    UserEventsEmitter.emit('user-created', {
        user: user,
    });
};
```

Figura 43: Excerto do código-fonte do *CreateUserCommandHandler*.

Na Figura 44 é demonstrado um exemplo da lógica de implementação do *UsersWriteModelRepository*, onde na coleção de utilizadores (*USERS*) é criado um documento, identificado pelo *id* do utilizador e cujo conteúdo são as restantes informações deste.

```
UsersWriteModelRepository.insertUser = async (user) => {
    const userData = User.fromModel(user);

    await firebase
        .child(USERS)
        .child(user.id)
        .set(userData)
        .catch((error) => {
            console.log('Error inserting user', error);
        });
};
```

Figura 44: Excerto do código-fonte do *UsersWriteModelRepository*.

De seguida é mostrada a correspondência entre cada uma das rotas, tanto para os recursos dos utilizadores como dos dispositivos, o correspondente comando ou query que será utilizado. As queries relativas ao recurso dos utilizadores devem tirar partido do repositório *UsersReadModelRepository*, enquanto que para os recursos dos dispositivos deverá ser utilizado o repositório *DevicesReadModelRepository*. Já para os comandos, a lógica é idêntica mas desta vez usando os repositórios *UsersWriteModelRepository* e *DevicesWriteModelRepository*, respectivamente.

Método HTTP	**Rota**	**Descrição**	**Comando/Query**
GET	/api/users	Obter a lista de utilizadores	GetUsersQuery
GET	/api/users/id	Obter a informação de um utilizador	GetUserByIdQuery
POST	/api/users/id	Adicionar um utilizador do sistema	CreateUserCommand
PUT	/api/users/id	Atualizar um utilizador do sistema	UpdateUserCommand
DELETE	/api/users/id	Apagar um utilizador do sistema	DeleteUserCommand

Tabela 3: Rotas para controlo de utilizadores com os respetivos comandos/queries.

Método HTTP	Rota	Descrição	Comando/Query
GET	/api/users/id/devices	Obter a lista de dispositivos do utilizador	GetDevicesQuery
GET	/api/users/id/devices/device_id	Obter a informação de um dispositivo do utilizador	GetDeviceByIdQuery
POST	/api/users/id/devices/device_id	Adicionar um dispositivo ao utilizador	CreateDeviceCommand
PUT	/api/users/id/devices/device_id	Atualizar um dispositivo do utilizador	UpdateDeviceCommand
DELETE	/api/users/id/devices/device_id	Apagar um dispositivo do utilizador	DeleteDeviceCommand

Tabela 4: Rotas para controlo de dispositivos com os respetivos comandos/queries.

Exemplo dos dados

Nesta secção serão demonstrados os exemplos de dados que esta aplicação gere nas bases de dados presentes na *Cloud Firestore*. Os dados armazenados na *Cloud Firestore* estão repartidos em duas coleções - *devices* e *users*, sendo a primeira relativa ao estado dos dispositivos, controlada através das intenções smart home e a segunda relativa a informações sobre os utilizadores e os dispositivos que estes mantêm, como tal controlada através da API desenvolvida.

```
{
    "led": {
        "OnOff": { "on": false }
    },
    "washer": {
        "Modes": { "load": "small" },
        "OnOff": { "on": true },
        "StartStop": {
            "isPaused": false,
            "isRunning": false
        },
        "Toggles": { "turbo": true }
    }
}
```

Listing 3: Exemplo da coleção que contém o estado dos dispositivos.

```
{
    "access_token": "fake_access_token",
    "devices": [
        {
            "deviceInfo": {
                "hwVersion": "1.0",
                "manufacturer": "Dinis Peixoto Raspberry Pi",
                "model": "led-pi",
                "swVersion": "1.0.1"
            },
            "id": "led",
            "name": {
                "defaultNames": [ "My Led" ],
                "name": "Led",
                "nicknames": [ "Led", "light", "led" ]
            },
            "traits": [ "action.devices.traits.OnOff" ],
            "type": "action.devices.types.LIGHT"
        },
    ],
    "email": "dinispeixoto@mail.com",
    "name": "Dinis Peixoto"
}
```

Listing 4: Exemplo da coleção que contém informação sobre os utilizadores e os seus dispositivos.

8.2.2 *Raspberry Pi*

A aplicação para refletir o estado dos dispositivos dos utilizadores em dispositivos físicos consiste num servidor web, utilizando a framework *Express*, pronto a receber pedidos provindos da *Smart home Action*. Ao receber pedidos, esta aplicação deverá ser responsável por ter a lógica necessária ao controlo dos periféricos conectados ao Raspberry Pi. A interação entre as aplicações smart home e esta é representada na Figura 45.

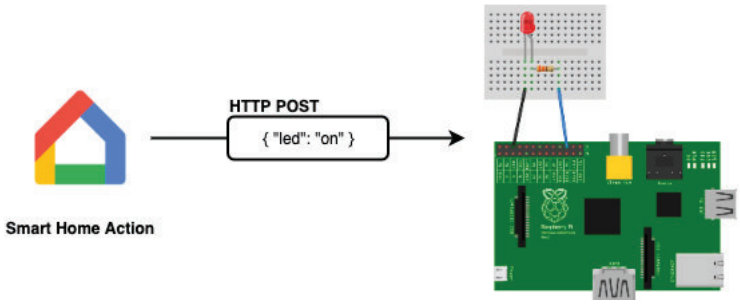


Figura 45: Diagrama da interação entre as aplicações smart home e Raspberry Pi.

Esta aplicação será a única que não estará alojada na *cloud*, mas sim num Raspbery Pi presente na casa do utilizador. Só desta forma é possível garantir uma interação próxima com os seus dispositivos físicos. Para simular estes dispositivos serão utilizados LEDs que devem ser ligados ou desligados conforme o estado dos dispositivos. Se um utilizador pedir para ligar a sua televisão inteligente, para os efeitos da presente dissertação, deveremos tornar isto visível através dos LEDs conectados como periféricos ao Raspberry.

```
var express = require('express');
var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO

var app = express();
var LED = new Gpio(23, 'out'); //use GPIO pin 4, and specify that it's for output

app.use(express.json());

app.post('/', (req, resp) => {
    if (req.body['led'] === 'on') {
        LED.writeSync(1);
        resp.send('Led is ON.');
    }
    else if (req.body['led'] === 'off') {
        LED.writeSync(0);
        resp.send('Led is OFF.')
    }
    else
        resp.send('Invalid command.')
});

app.listen(3000);
```

Figura 46: Excerto do código-fonte da aplicação do Raspberry Pi.

Na Figura 46 temos um exemplo de uma aplicação que, conforme os pedidos recebidos, irás atualizar o estado de um dos seus periféricos. Os pins GPIO[7] no Raspberry Pi, apresentados na Figura 47, permitem a criação de circuitos, será através destes que será enviada corrente para ligar o LED físico. No exemplo anterior podemos verificar que quando a aplicação recebe um pedido para ligar o LED, é utilizado o GPIO 23 para emitir corrente, ligando o periférico a este conectado.

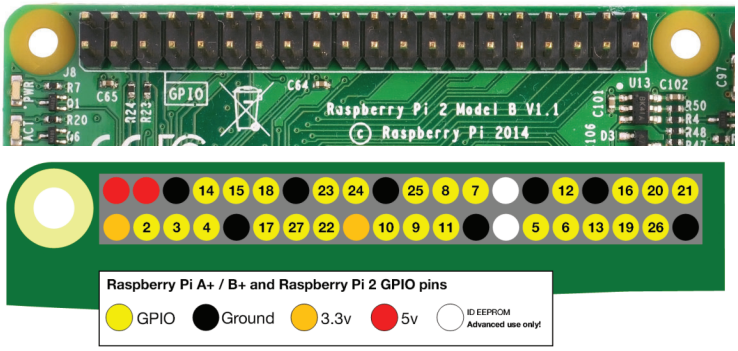


Figura 47: Correspondência dos pins no Raspberry Pi (figura de cima) com o seu tipo (figura de baixo).

Para esta aplicação estar visível para a aplicação smart home foi utilizada a ferramenta Localtunnel[8] que gera um URL dinâmico para o servidor web em execução no Raspberry Pi que é depois utilizado pela aplicação de smart home para realizar os pedidos para controlo dos dispositivos.

8.2.3 *Conversational Action*

A implementação da *Conversational Action* tem a particularidade de incluir o desenvolvimento do módulo de *Natural Language Processing* através da plataforma Dialogflow. As mensagens dos utilizadores são primeiro processadas por este módulo e só depois é utilizado o *fulfillment* para aplicar a lógica necessária, que neste caso será tirar partido da *Smart home Action* para controlar informação e estado dos dispositivos inteligentes do utilizador. Esta será a aplicação com que o utilizador irá interagir diretamente, funcionando como o *front-end* de todo o sistema até então desenvolvido.

7 https://en.wikipedia.org/wiki/General-purpose_input/output
8 https://localtunnel.github.io/www/

Configurações na plataforma Actions on Google

Antes de começar o desenvolvimento propriamente dito da aplicação é necessário fazer a configuração inicial na plataforma *Actions on Google*. Esta configuração é um pouco distinta da *Smart home Action* uma vez que a ação será especificada através da sua instância no Dialogflow, tal como podemos ver na Figura 48. Isto garante que as invocações dos utilizadores sejam automaticamente processadas pelo Dialogflow, devido à integração existente entre ambas as plataformas.

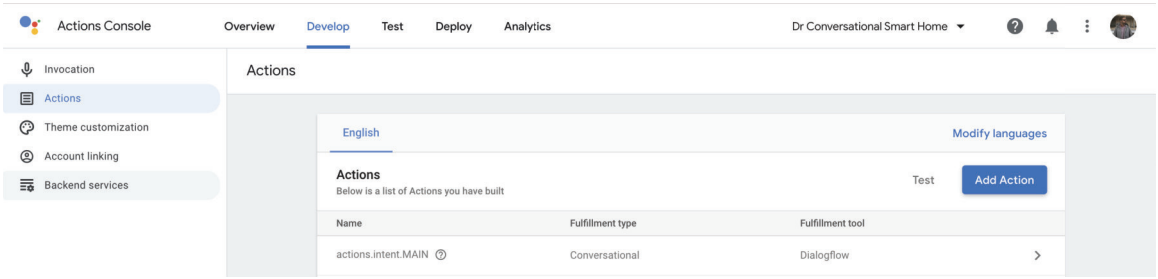


Figura 48: Configuração da ação de conversação na plataforma *Actions on Google*.

Módulo de Natural Language Processing

Módulo responsável pela tradução das mensagens do utilizador em dados prontos a ser processados pelo *fulfillment*, isto é o *back-end* da aplicação. O foco desta secção será descrever o que foi desenvolvido na plataforma Dialogflow para fazer este processamento da mensagem, distribuindo-o por intenções e entidades.

- **Intenções**

  Cada funcionalidade da aplicação deverá ser traduzida numa intenção. Enquanto que para intenções utilitárias, como apagar os dados da sessão de conversação do utilizador ou obter as funcionalidades de um dispositivo geram intenções únicas, cada uma das operações para interagir com o estado dos dispositivos deverá variar conforme o *trait* para o qual a operação se irá refletir, dando origem a múltiplas intenções diferentes.

  Desta forma, para uma melhor organização durante a criação das intenções de smart home disponíveis para utilização, seguiu-se a seguinte nomenclatura: `smarthome.device.trait.command.smarthome_intent`. Como exemplo, para a intenção correspondente à execução de uma operação Execute para o *trait* OnOff e comando OnOff teremos a seguinte intenção: `smarthome.device.onoff.onoff.execute`, ou seja a intenção reconhecida pelo NLP quando um utilizador pretende desligar ou ligar um determinado dispositivo.

Por sua vez, caso o utilizador pretenda agendar uma ação, a intenção será `smarthome.device.onoff.onoff.schedule`.

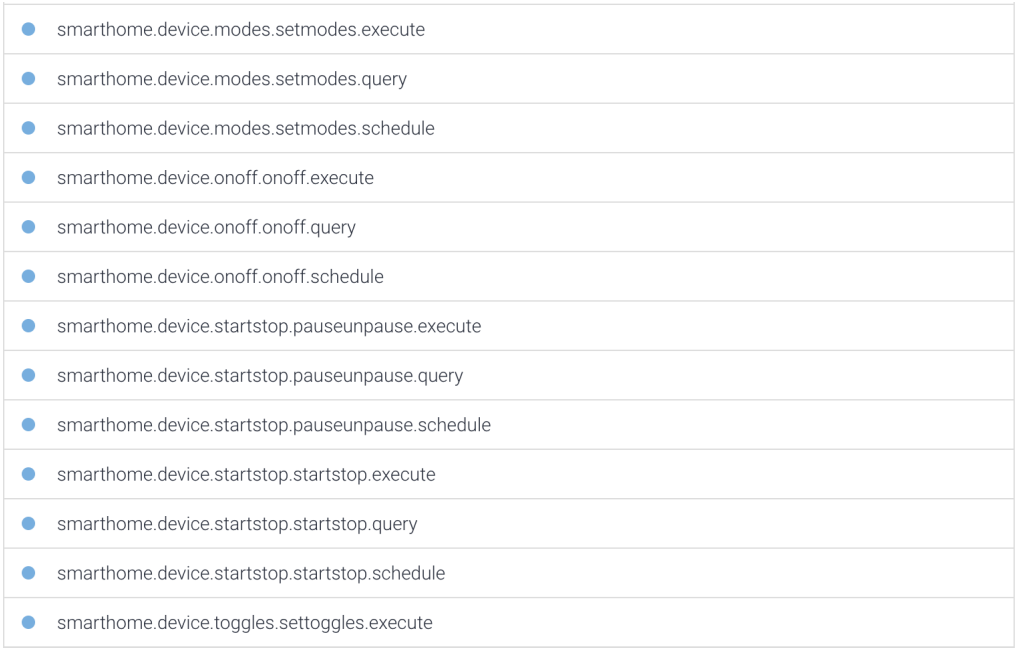


Figura 49: Exemplo de algumas das intenções disponíveis no Dialogflow.

No momento de criação de uma intenção, é necessário fornecer três componentes:

- *Training phrases*: exemplos de input do utilizador;
- *Action and parameters*: entidades que possam estar presentes nas frases exemplo fornecidas;
- *Response* ou *Fulfillment*: resposta estática ou indicação de um *fulfillment* para esta.

Como sabemos para as intenções será sempre utilizado um *fulfillment*, correspondendo à aplicação de conversação a desenvolver, permitindo respostas dinâmicas para cada um dos pedidos recebidos.

Para facilitar a compreensão da criação de intenções mostraremos a sua criação através de um exemplo. Considerando como exemplo o *trait* OnOff, devem ser criadas três intenções: Query, Execute e Schedule; estas permitem consultar, alterar e agendar uma ação para alterar o seu estado, respetivamente. Para não tornar o exemplo demasiado extenso, o foco será direcionado unicamente para a intenção de Execute.

Relativamente às frases exemplo ou *training phrases*, devem ser especificados o maior número de frases possíveis de modo a que o modelo NLP gerado depois de treinado com estas obtenha melhores resultados. É de notar que o módulo de NLP será capaz de reconhecer frases semelhantes e não só as que forem explicitamente fornecidas.

Ao fornecer exemplos de frases são automaticamente detetadas as entidades presentes (se estas estiverem previamente criadas). Na figura 50 são demonstrados alguns exemplos de frases fornecidas, embora existam muitos mais dos que os que estão presentes na figura.

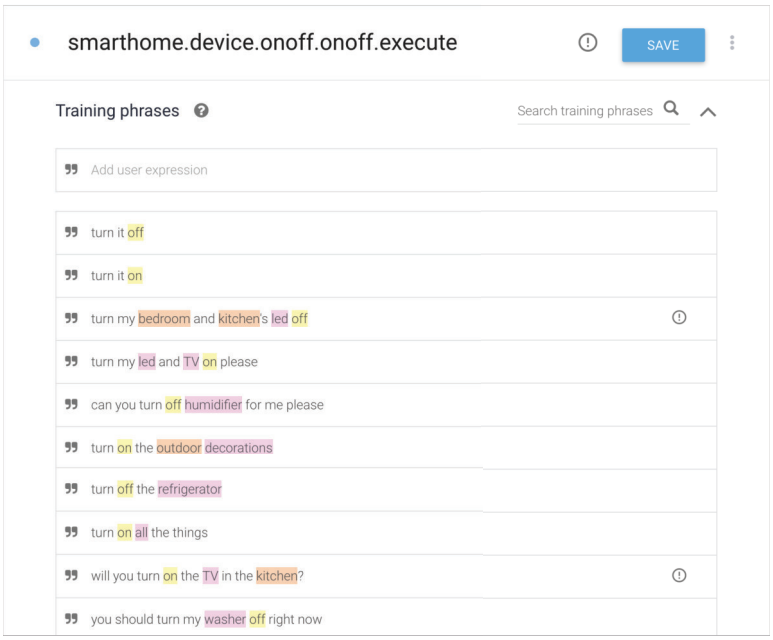


Figura 50: Exemplo de *trainig phrases* para a intenção.

Na secção *Action and parameters* devem estar presentes informações sobre as entidades utilizadas nas frases exemplo. Para o exemplo considerado serão necessárias, pelo menos, três entidades:

- command_on_off: comando que indica se é para ligar ou desligar o dispositivo;
- devices: lista de dispositivos para os quais a operação deverá ser aplicada;
- room: entidade opcional que descreve as divisões da casa onde os dispositivos se encontram;

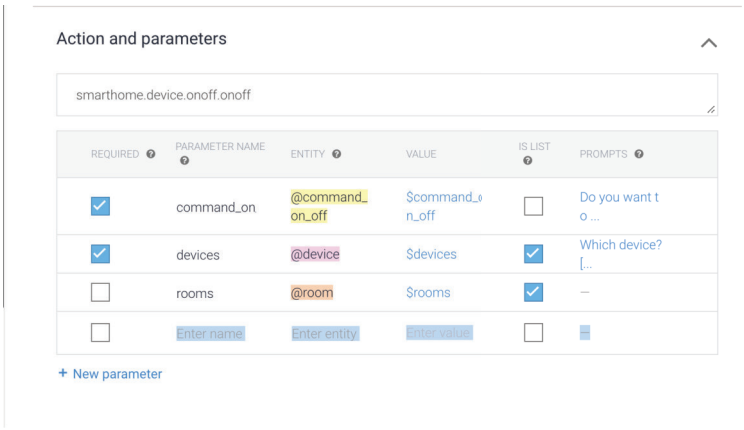


Figura 51: Exemplo de *Action and parameters* para a intenção.

Por fim, deverá ser utilizado um *fulfillment* para fazer o processamento desta intenção, pelo que o agente do Dialogflow deverá ser devidamente configurado com o URL do *fulfillment*, isto é, da aplicação conversacional que ficará alojada nas *Cloud Functions*.

Figura 52: Exemplo da definição do URL do *fulfillment*.

- **Entidades**

  De seguida apresentam-se algumas das entidades criadas, sendo que todas estas são utilizadas pelas intenções anteriormente referidas e não necessariamente no exemplo fornecido. É de notar que as entidades com o prefixo *sys* são padrão e disponibilizadas pela plataforma Dialogflow (e.g. *sys.time* e *sys.date*).

  Para cada comando disponível será necessária a correspondente entidade identificando os possíveis valores que esse mesmo comando poderá apresentar. O comando OnOff apresentado no exemplo poderá ter os valores ON ou OFF. Como exceções temos as entidades de *Modes* e *Toggles* que poderão ter valores customizados pelo utilizador já que é precisamente este o seu propósito.

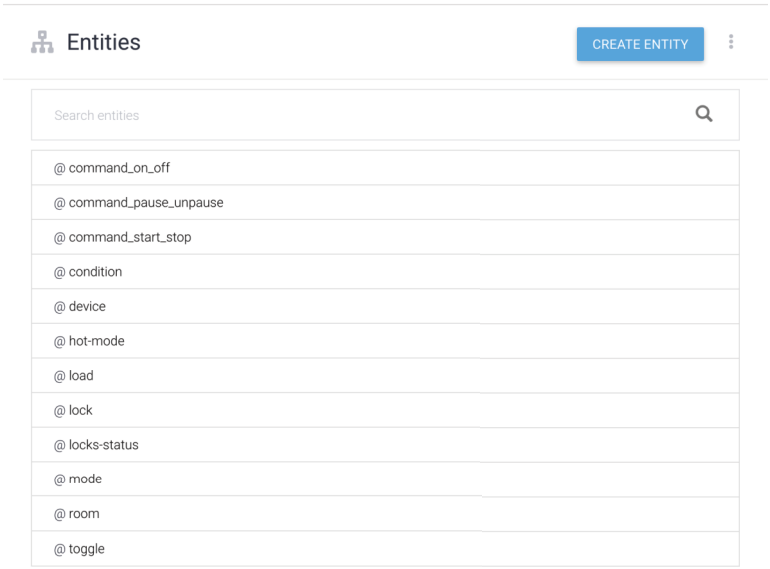


Figura 53: Exemplo das entidades criadas para utilização nas intenções.

Considerando o exemplo demonstrado anteriormente, eis a definição das entidades *command§_on_off* e *devices*, as Figuras 54 e 55 respetivamente, utilizadas na intenção de Execute do trait e comando OnOff.

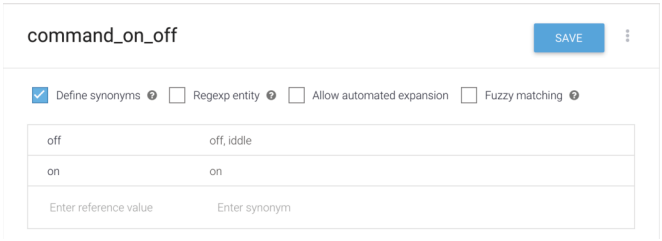


Figura 54: Exemplo da definição da entidade *command_on_off*.

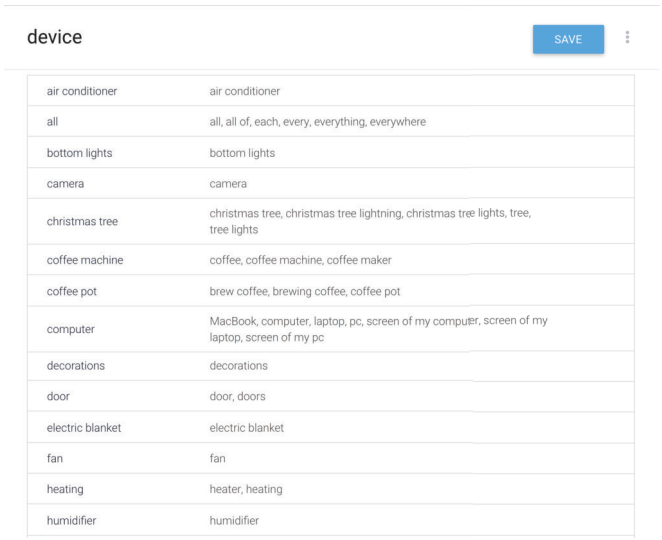


Figura 55: Exemplo da definição da entidade *device*.

Fulfillment

Depois da invocação do utilizador ser processada pelo agente do Dialogflow é realizado um pedido ao *fulfillment* da ação, sendo este o componente da lógica de negócio da aplicação conversacional. Esta aplicação deve seguir com a arquitetura previamente estabelecida, focada em três componentes principais: os módulos NLP, responsáveis pela comunicação com a plataforma Dialogflow; os controladores, onde a lógica para cada uma das intenções é aplicada; e por fim os *Message Builders* cujo propósito é a criação das respostas para o utilizador, conforme o resultado do processamento anterior.

Para explicar a implementação de cada um destes componentes, seguiremos com o mesmo exemplo anterior, isto é para o trait OnOff. Pela figura 56 podemos observar que os *handlers* recebem o estado atual da conversação através da variável `conv`, assim como cada uma das entidades especificadas na plataforma Dialogflow (`command_on_off`, `devices` e `rooms`). Estes *handlers* têm lógicas um pouco distintas para as diferentes intenções (Execute, Query e Schedule), com especial relevância para a última, onde é feita a verificação da subscrição do utilizador para as notificações, já que uma será enviada caso este pretenda agendar uma determinada operação.

```
OnOffNlp.Execute = (conv, { command_on_off, devices, rooms }) => {
    SmartHomeController.handleExecute(TRAITS.ON_OFF.name, COMMANDS.ON_OFF.name, command_on_off, devices);

    const response = OnOffMessageBuilder.buildMessageForExecute(devices, command_on_off);
    conv.ask(response);
};

OnOffNlp.Query = async (conv, { command_on_off, devices, rooms }) => {
    const data = await SmartHomeController.handleQuery(devices);

    const response = OnOffMessageBuilder.buildMessageForQuery(devices[0], data);
    conv.ask(response);
};

OnOffNlp.Schedule = (conv, { command_on_off, devices, rooms, date, time }) => {
    if (!conv.user.storage.push_notification_permission) {
        conv.ask('Update permission for setting up push notifications');
        conv.ask(new UpdatePermission({ intent: 'push.notification.info' }));
        conv.user.storage.push_notification_permission = true;
    }

    SmartHomeController.handleSchedule(TRAITS.ON_OFF.name, COMMANDS.ON_OFF.name, command_on_off, devices, date, time);

    const response = OnOffMessageBuilder.buildMessageForSchedule(devices[0], command_on_off, buildDateTime(date, time));
    conv.ask(response);
};
```

Figura 56: Excerto do código-fonte de OnOffNlp.

Nos controladores a lógica necessária será realizar um pedido à *Smart home Action*, conforme as mensagens do utilizador. Conforme o resultado deste pedido deverá ser gerada uma mensagem a enviar ao utilizador. Na figura 57 temos um exemplo de um *Message Builder* responsável por criar mensagens para as diferentes intenções disponíveis.

```
OnOffMessageBuilder.buildMessageForExecute = (device, command) => {
    return `Turning your ${device} ${command}...`;
};

OnOffMessageBuilder.buildMessageForQuery = (device, data) => {
    return `Your ${device} is ${data.payload.devices[device].on ? 'on' : 'off'}.`;
};

OnOffMessageBuilder.buildMessageForSchedule = (device, command, datetime) => {
    return  `I'm going to turn your ${device} ${command}` +
            ` on ${datetime.format('dddd, MMMM Do YYYY')}` +
            ` at ${datetime.format('h:mm:ss a')}...`;
};
```

Figura 57: Excerto do código-fonte de OnOffMessageBuilder.

Notificações

Quando um utilizador realiza uma invocação correspondente a uma intenção de Schedule tem de ser agendada uma operação de Execute num determinado dispositivo, o que exige a utilização de um *scheduler*. Através da utilização de um *scheduler* é possível agendar *jobs* que são executados numa determinada data ou até recorrentemente. A data para o qual os

jobs serão executados deverá ser uma das entidades reconhecidas no módulo de NLP para este tipo de intenções.

O problema que surge com a utilização de um *scheduler* é que este só funciona se a aplicação tiver um estado e este seja partilhado durante todas as invocações à mesma, o que, segundo vimos em 7.2, não acontece quando desenvolvemos uma *serverless architecture*, através da utilização de *Cloud Functions*. Tendo em conta este problema, foi necessária a criação de uma aplicação adicional - *Smart home Scheduler* - com o único propósito de gerir o agendamento da execução de operações sob os dispositivos. Esta aplicação não pode dispor uma arquitetura *serverless* e como tal ficou alojada numa plataforma diferente.

A única lógica que precisa do conceito de estado é o agendamento de *jobs* para execução no futuro. Como tal, a restante lógica necessária para o correto funcionamento das *push notifications*[9] deverá estar presente na *Conversational Action*. Esta lógica inclui persistir as subscrições dos utilizadores numa base de dados, utilizando a *Cloud Firestore* e a devida verificação desta subscrição aquando invocação de intenções de Schedule.

8.2.4 *Aplicação Smart Home Scheduler*

Das aplicações implementadas, esta será a única que precisa efetivamente de ter um estado e que este seja partilhado durante as diversas invocações à mesma. Isto leva a que, ao contrário das restantes aplicações na *cloud*, isto é, já excluindo a aplicação do Raspberry Pi, esta não seja implementada através da utilização de *Cloud Functions*.

Esta aplicação embora também esteja alojada na *cloud*, não estará na Google Cloud mas sim no Heroku[10]. O Heroku permite, de forma gratuita, alojar aplicações em servidores na *cloud* que ficam continuamente disponíveis para os seus utilizadores. Estas aplicações, ao contrário do que acontece com as *Cloud Functions*, não são instanciadas de cada vez que recebem um pedido, o que garante que os *jobs* agendados pelo *scheduler* não sejam perdidos.

9 https://developers.google.com/actions/assistant/updates/notifications
10 https://www.heroku.com/

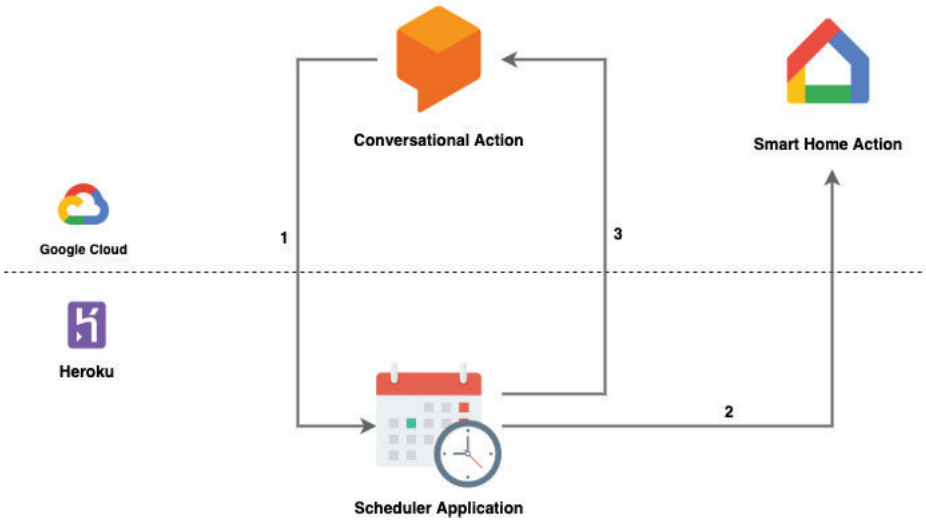


Figura 58: Diagrama da integração das aplicações: Conversational Action, Scheduler Application e Smart home Action.

Pela figura 58 podemos facilmente verificar o fluxo entre as aplicações existentes durante uma intenção de Schedule. Assim que a *Conversational Action* recebe um pedido do utilizador para agendar uma determinada operação é enviado um pedido à aplicação Scheduler, representado pelo ponto 1 no diagrama. Este pedido contém a data e hora para a qual o utilizador pretende agendar a operação, informações sobre o pedido (URL e payload) que deverá ser realizado quando o momento de realizar a operação é alcançado e ainda informação sobre a notificação a enviar ao utilizador.

```
{
    "date": "2019-07-25T21:06:10.690Z",
    "request": {
        "url": "https://example/smarthome",
        "payload": (...)
    },
    "notification": {
        "user": 1,
        "data": { "title": "Your led changed its state" }
    }
}
```

Listing 5: Exemplo da informação contida no pedido à aplicação *Scheduler*.

Assim que a aplicação *Scheduler* recebe este pedido cria um *job* para execução para a data e hora contida no mesmo. Quando o momento de execução é alcançado, o *job* é espoletado e deverá ser responsável por duas invocações.

A primeira invocação, correspondente ao ponto 2 do diagrama, será utilizada conforme a informação contida no pedido inicialmente recebido. Neste caso, o URL deverá ser o da *Smart home Action* e o payload correspondente ao de uma intenção de Execute.

Assim que este pedido é realizado com sucesso, garantimos que o estado do dispositivo já foi alterado e resta apenas enviar uma notificação ao utilizador. Com base na informação da notificação provinda do pedido inicial é enviado um pedido à *Conversational Action* para uma *Cloud Function* própria, segundo a rota /notifications com a informação da notificação a enviar ao utilizador.

A comunicação entre estas aplicações é realizada por HTTP, o que num cenário real facilmente se relevaria uma má abordagem. Dada a natureza assíncrona dos pedidos que estão a ser realizados entre as aplicações para o agendamento de operações nos dispositivos do utilizador, num cenário real seria mais vantajosa a utilização de um sistema de mensagens como Kafka[11] ou RabbitMQ[12]. A utilização de um sistema de mensagens, por exemplo com uma abordagem *Publisher/Subscriber* iria permitir um maior desacoplamento entre as aplicações, aumentando a escalabilidade do sistema como um todo.

8.3 CASOS DE ESTUDO

Para demonstrar que os requisitos foram devidamente implementados serão apresentados os casos de estudo iniciais que levaram à implementação de funcionalidades adicionais ao Google Assistant. A concretização destes casos de estudo é importante já que revela que o sistema desenvolvido e as aplicações que o compõem permitem, em sintonia, o devido funcionamento dos requisitos inicialmente apresentados. Para cada um dos casos de estudo será apresentado um diagrama de sequência da interação entre as aplicações e ainda uma demonstração dos mesmos na prática, isto é, numa utilização real com o assistente.

8.3.1 *Identificação de quais as funcionalidades de um determinado dispositivo*

Enquanto utilizador do Google Assistant e ao longo do desenvolvimento de uma aplicação smart home para o mesmo, uma das funcionalidades que senti mais necessidade foi a possibilidade de perguntar ao assistente quais as funcionalidades (ou *traits*) que um determinado dispositivo apresenta, tornando-se por vezes complicado descobrir qual a gama

11 https://kafka.apache.org/
12 https://www.rabbitmq.com/

de comandos que é possível utilizar para um determinado dispositivo. Desta forma, uma das funcionalidades que fará parte dos casos de estudo apresentados nesta secção será a possibilidade da invocação por parte dos utilizadores do tipo *Hey Google, could you tell me my led's traits?* ou *Hey Google, what can I do with my led?*.

A adição desta funcionalidade exigiu algumas alterações apenas ao nível da aplicação conversacional, já que a aplicação smart home incluía já a API que permite fazer consulta das informações dos dispositivos de um determinado utilizador. As alterações exigidas na aplicação conversacional começam no seu módulo de NLP, foi necessária a criação de uma intenção para permitir a utilização desta funcionalidade aos utilizadores. Já na própria aplicação, foi necessária a adição de lógica para fazer o devido pedido à API da aplicação smart home.

Na Figura 59 é apresentado um diagrama de sequência descritivo da integração entre as aplicações do sistema para concretizar o pedido inicial do utilizador.

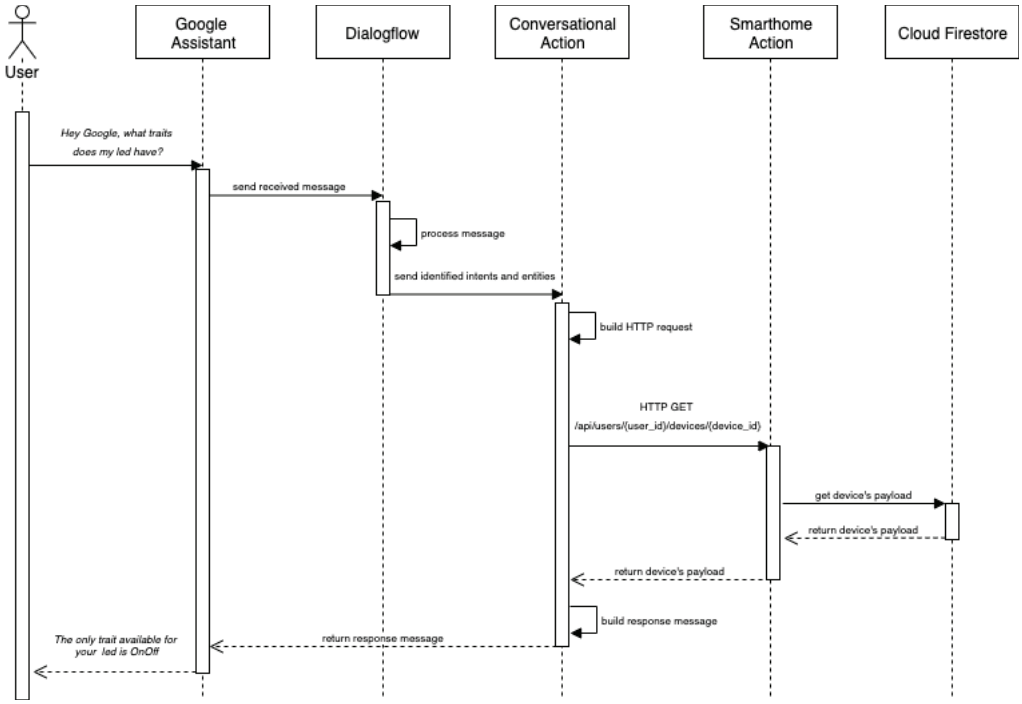


Figura 59: Diagrama de sequência da identificação de funcionalidades de um dispositivo.

Demonstração

Na Figura 60 é apresentada uma situação real de um utilizador a tirar partido desta funcionalidade através da aplicação conversacional, o *Dr. Smarthome*. A mensagem de resposta da aplicação é meramente ilustrativa pelo que, numa situação real, deveria ser explorada

para melhorar a experiência do utilizador (e.g. *Oh, looks like you can only turn it on or off*). Assim sendo, embora a mensagem apresentada tenha literalmente os *traits* do dispositivo, esta poderia ser explorada de diferentes maneiras de modo a enriquecer a experiência do utilizador com o assistente.

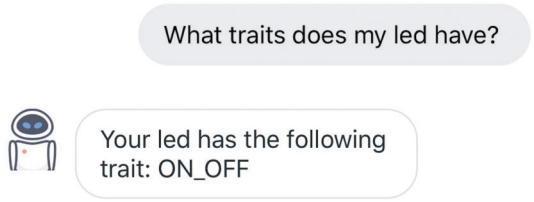


Figura 60: Demonstração da identificação de funcionalidades de um dispositivo.

8.3.2 *Agendamento de uma determinada operação*

Uma outra funcionalidade em falta no assistente é a possibilidade de agendar comandos para mudar o estado dos dispositivos. Atualmente o Google Assistant já permite que algumas ações sejam agendadas através da utilização de Routines[13]. No entanto não existe a possibilidade de o fazer através do assistente, por via da voz. Após contacto com o próprio suporte da Google sobre esta funcionalidade, a resposta foi que ainda não teriam pensado sobre a possibilidade de realizar estas operações via voz, e que a ideia seria transmitida à equipa de produto e/ou engenharia.

Como o assistente não tem esta funcionalidade por enquanto, faz sentido que a mesma seja explorada como um caso de estudo. Durante a especificação e até implementação das aplicações ao longo do presente documento já foram explicadas as alterações necessárias na aplicação conversacional com a introdução das intenções de *Schedule* e a introdução da própria aplicação Scheduler cujo o único propósito no sistema será o processamento de invocações do utilizador relativas a esta funcionalidade.

Na Figura 61 é apresentado um diagrama de sequência descritivo da integração entre as aplicações do sistema até ao próprio dispositivo físico, para satisfazer o pedido inicial do utilizador.

13 https://support.google.com/googlehome/answer/7029585?co=GENIE.Platform%3DAndroid&hl=en

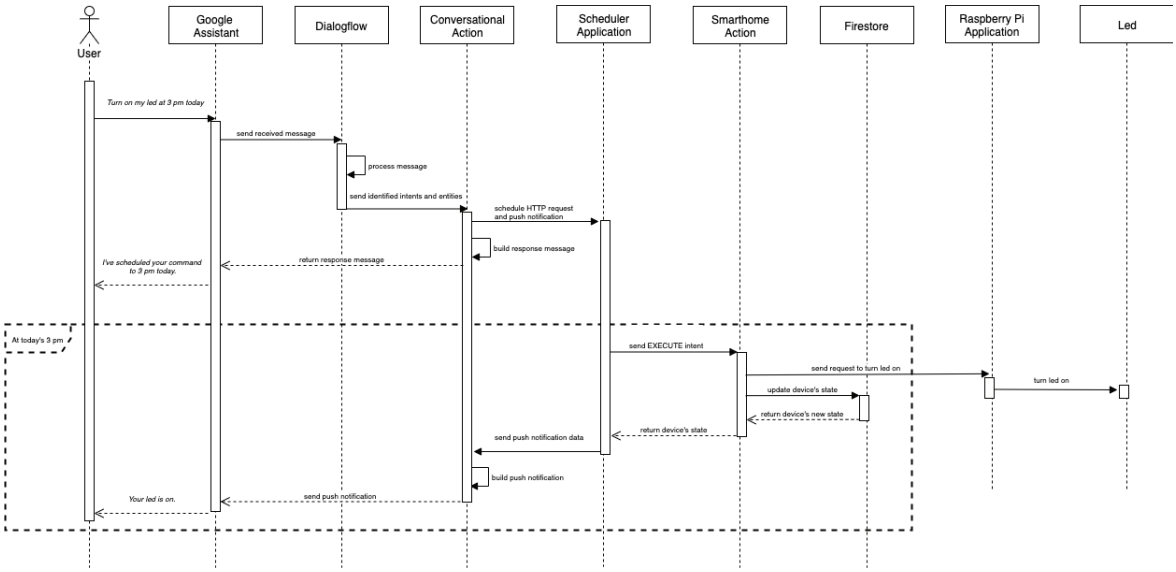


Figura 61: Diagrama de sequência do agendamento de uma determinada operação.

Demonstração

Esta funcionalidade exige o envio de uma *push notification* para o utilizador assim que a operação agendada é realizada, mudando o estado do dispositivo. Nas figuras que se seguem vemos, à esquerda, um exemplo de uma situação real onde um utilizador agenda uma determinada operação e, à direita, a receção da notificação, alertando-o que o estado do seu dispositivo foi alterado conforme requisitado.

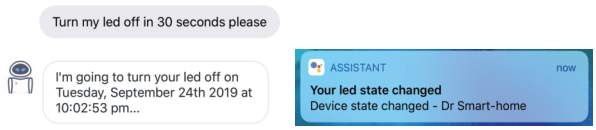


Figura 62: Demonstração do agendamento de uma operação (à esquerda) e notificação resultante quando a operação é realizada (à direita).

8.4 DESAFIOS

Nesta secção o objetivo será enumerar e detalhar os diversos desafios que surgiram aquando implementação do sistema, seguindo as arquiteturas apresentadas no capítulo anterior. Embora as arquiteturas idealizadas garantam um sistema mais escalável, modular e robusto, a implementação das aplicações de acordo com estas arquiteturas deu origem a desafios acrescidos, que serão de seguida explicados.

8.4.1 *Arquitetura Serverless*

A utilização de uma arquitetura *serverless* faz com que as aplicações desenvolvidas não tenham estado, já que é impossível que este seja partilhado entre as invocações realizadas às diversas funções que as constituem. Nas aplicações para os assistentes o estado da conversação é, regra geral, garantido pela própria plataforma de NLP, neste caso o Dialogflow, no entanto podem existir situações em que seja necessário guardar o estado correspondente à lógica de processamento dos pedidos dos utilizadores.

Durante o desenvolvimento foi possível verificar esta necessidade na funcionalidade de agendamento de operações. Para conseguir garantir o seu correcto funcionamento foi necessário desenvolver uma aplicação adicional para gerir as operações agendadas pelos utilizadores, sendo que esta aplicação ficou até alojada numa plataforma distinta, onde está em constante execução, ao contrário do que acontece com as funções desenvolvidas resultantes da arquitetura *serverless* utilizada.

Outro dos desafios originados com o desenvolvimento de aplicações segundo este tipo de arquiteturas é o próprio desenvolvimento local, isto é, antes das aplicações serem efetivamente enviadas para a *cloud*. Não é possível simular localmente o comportamento que estas funções têm quando estão na *cloud*, dando origem a um desafio acrescido aquando desenvolvimento das mesmas.

8.4.2 *Command-Query Responsibility Segregation*

A introdução do padrão arquitetural CQRS introduz também muita complexidade ao nível do código aplicacional, sendo que deve ser avaliado se os ganhos com a inclusão deste padrão se sobrepõe à complexidade resultante.

Enquanto que para a aplicação desenvolvida (aplicação smart home) foram usados apenas dois modelos - Utilizadores e Dispositivos - à medida que o número de modelos fosse aumentando a dificuldade de gestão do código da aplicação iria também aumentar, já que seria necessário ter os modelos replicados para os sistemas de escrita e leitura. Esta replicação entre os modelos dos dois sistemas exige que alterações num dos modelos seja também refletida no outro, deixando a aplicação mais exposta a possíveis erros, por exemplo.

No decorrer do desenvolvimento, o mecanismo utilizado para fazer a sincronização entre os modelos de escrita e leitura foi através de eventos nativos de JavaScript. Numa situação real deveria ser utilizada uma tecnologia mais adequada para o efeito, por exemplo através de mensagens. A introdução de uma destas tecnologias para o sistema de sincronização, gera complexidade no código aplicacional. Para além disto, uma falha neste mecanismo de sincronização dá origem a talvez um dos maiores desafios da utilização do *CQRS* - a

inconsistência entre os dois sistemas. No sistema desenvolvido a quantidade de dados utilizada é relativamente reduzida, pelo que em situações de falha na sincronização dos dois modelos não foi complicado garantir que existisse consistência novamente. No entanto, para um caso real, uma situação destas exigiria correr uma migração para garantir que a inconsistência gerada pela falha deste mecanismo fosse resolvida, sendo que esta migração poderia abranger milhares ou até milhões de dados relativos aos utilizadores e seus dispositivos.

Desta forma, a introdução deste padrão deve ser ponderada. Devem ser exploradas os benefícios de uma arquitetura que tire partido do mesmo, comparando-os com as limitações e desafios acrescidos que podem surgir.

8.4.3 *Arquitetura da aplicação smart home*

A arquitetura da aplicação smart home é, em grande parte, definida pela utilização do padrão CQRS, pelo que os desafios abordados anteriormente descrevem parte dos que surgiram durante a implementação desta.

Enquanto que os objetivos com ambas as aplicações era serem agnósticas dos assistentes digitais que seriam utilizados para destas tirarem partido, isto revelou-se um dos maiores desafios durante o desenvolvimento da mesma. A existência de serviços para gerir tipos de dispositivos (*DevicesService*) e tipos de funcionalidades (*TraitsService*) é muito dependente do ecossistema e conceitos utilizados no Google Assistant, pelo que a utilização desta aplicação noutro assistente, como a Alexa, poderia exigir algumas alterações ao nível destes serviços.

A comunicação com o Raspberry Pi, ou seja, o desenvolvimento de *gateways* para os dispositivos foi também um dos grandes desafios ao pôr em prática esta arquitetura. Para o sistema desenvolvido para a presente dissertação foi utilizada apenas comunicação através de pedidos HTTP. No entanto, numa situação real, os mecanismos de comunicação poderiam ser através de mensagens ou RPC, para garantir mais flexibilidade com o tipo de dispositivos com os quais é possível interagir.

8.4.4 *Arquitetura da aplicação conversacional*

A arquitetura proposta para a aplicação conversacional tem como foco permitir que esta aplicação consiga ser partilhada por diversos assistentes digitais para tirar partido da aplicação smart home. Isto constituí o principal desafio do desenvolvimento desta arquitetura.

Os módulos de NLP desenvolvidos fazem correspondência com a plataforma NLP utilizada, neste caso o Dialogflow. Esta plataforma poderia ser utilizada para fazer o pro-

cessamento de mensagens provindas de utilizadores de diferentes assistentes digitais, no entanto a lógica para lidar com cada uma das intenções poderia ser diferente entre os assistentes, isto é, a mesma intenção poderia resultar num pedido a um serviço externo diferente dependendo do assistente digital utilizado. Esta lógica teria de ser introduzida nestes componentes, criando alguma complexidade no código aplicacional.

O mesmo acontece com a coesão entre a camada de processamento de intenções - os Controladores - e a camada que constrói as respostas para o utilizador - os *Message Builders*. Enquanto que o propósito dos *Message Builders* é, precisamente, adpatar diferentes tipos de respostas para utilizadores provindos de diferentes tipos de assistentes, a coesão entre a camada de processamento e a utilização destes *message builders* iria necessitar de lógica adicional para, conforme o pedido inicial do utilizador, utilizar o *message builder* correcto para lhe enviar a resposta, adicionando, tal como no caso anterior, alguma complexidade à aplicação.

9

CONCLUSÃO E TRABALHO FUTURO

O facto do desenvolvimento de aplicações para assistentes digitais ser uma prática tão recente leva a que ainda não existam convenções e padrões arquiteturais bem definidos. Isto aliado ao facto de grande parte das aplicações disponíveis não serem abertas para a comunidade, acresce a dificuldade tanto ao nível de projeção de soluções arquiteturais como do próprio desenvolvimento destas aplicações.

Apesar destas dificuldades, os objetivos delineados para o desenvolvimento da presente dissertação foram alcançados, isto é, foram delineadas arquiteturas para dois tipos de aplicações (de conversação e smart home) e ainda conseguida uma solução para controlo de dispositivos IoT através de um dos diversos assistentes digitais existentes, neste caso o Google Assistant. O sistema desenvolvido permite ainda contornar a falta de flexibilidade imposta pelas soluções atualmente existentes, na medida em que permite a utilização de quaisquer tipo de invocações, sem qualquer restrição sob as mesmas.

Ultrapassada esta falta de flexibilidade imposta por outras soluções foram ainda apresentados casos de estudo que põem a utilização de invocações flexíveis em prática, sendo esta demonstrada através da inclusão de novas funcionalidades relativas ao controlo de dispositivos, como por exemplo o agendamento de operações nos mesmos.

Durante o desenvolvimento desta solução foram respeitadas as diversas condicionantes para garantir uma arquitetura escalável. A existência de uma arquitetura escalável na solução apresentada foi de extrema importância para que o desenvolvimento futuro da aplicação e sua arquitetura não sejam comprometidos com o aumento do número de utilizadores e respectivos dispositivos que necessitem de ser geridos através da solução final.

9.1 EXPERIÊNCIA OBTIDA

Durante o desenvolvimento da solução final foi necessário o contacto com muitas tecnologias novas e metodologias de desenvolvimento diferentes.

Começando pela linguagem de programação JavaScript, em particular do *runtime environment* Node. Apesar da curta curva de aprendizagem desta linguagem, devido à sua natureza maioritariamente assíncrona, foi necessário criar uma mentalidade de desenvolvimento um pouco distinta da que estava habituado.

Desenvolver aplicações para utilização por voz exige conhecimentos diferentes relativos ao *front-end* das aplicações desenvolvidas. Enquanto que numa aplicação visual seria utilizada uma framework para desenvolvimento como *React*, *Angular* ou *Vue*, nestas aplicações o desenvolvimento é muito particular e exige, regra geral, a utilização de uma plataforma responsável pelo processamento das mensagens dos utilizadores. A plataforma utilizada foi o Dialogflow e exigiu muito tempo na sua documentação para explorar a melhor forma de tirar partido das suas funcionalidades.

O facto de praticamente todas as aplicações estarem na cloud e todos os testes sobre as mesmas terem de ser realizados sobre estas condições, exigiu que fossem exploradas as diferentes ferramentas para debug, monitorização e estatísticas, oferecidas pela infraestrutura utilizada - neste caso o Google Cloud. Problemas relativos a interoperabilidade das aplicações e plataformas (e.g. Dialogflow) e até permissões sobre a utilização de cada uma destas foram recorrentes, pelo que foi adquirida experiência neste sentido.

A procura pela escalabilidade do sistema obrigou o estudo de padrões arquitecturais que fossem utilizados no mundo real e tivessem mais-valias confirmadas. Em particular, a investigação sobre *CQS/CQRS* e *Serverless architecture*, em conjunto com outros padrões que foram também investigados mas que acabaram por não ser aplicados. Ainda com o foco direccionado para a arquitetura, para cada uma das aplicações foi desenvolvida uma arquitetura modular para que a necessidade de introdução de novos componentes não tivesse qualquer impacto sobre a arquitetura existente, por exemplo a disponibilização de um novo tipo de funcionalidade por parte do Google Assistant, exigiria apenas a criação de um componente novo em cada uma das aplicações, conversacional e smart home.

9.2 Trabalho futuro

Embora os objetivos estejam efetivamente cumpridos, existem muitas possibilidades de melhoria na solução final conseguida e ainda alguns pontos que podiam ser explorados nesta para no futuro conseguir oferecer uma solução mais completa, isto é, com mais funcionalidades para os seus utilizadores.

Relativamente à arquitetura do sistema, grande parte da comunicação entre as aplicações é realizada através de pedidos HTTP. Existem, no entanto, situações onde poderia ser explorada a utilização de mensagens para fazer a comunicação das aplicações, devido à natureza assíncrona que existe na comunicação das mesmas - um exemplo disto seria a comunicação

da aplicação de conversação com a aplicação de schedule, em que não existe a necessidade da utilização de pedidos síncronos. A migração de alguns pedidos HTTP para mensagens, através da utilização de um serviço como RabbitMQ[1] ou Kafka[2] iria garantir não só um maior desacoplamento das aplicações como também melhorias ao nível da escalabilidade das mesmas. Poderiam ser explorados protocolos como MQTT (*Message Queuing Telemetry Transport*) e CoAP (*Constrained Application Protocol*), mais comuns em cenários de desenvolvimento dentro do paradigma dos dispositivos IoT[8], o primeiro devido à comunicação orientada a mensagens, segundo um padrão *publish-subscribe* baseado em TCP, e o segundo, muito semelhante ao HTTP mas baseado em UDP e otimizado para ambientes de recursos computacionais escassos, sendo isto verificado pela sua utilização maioritária em ambientes constituídos por dispositivos IoT.

A arquitetura proposta é um pouco enviesada pelo assistente digital utilizado, isto é, o Google Assistant e ainda pelos use-cases que foram implementados para acréscimo de funcionalidades. Isto poderia vir a revelar-se um factor crucial para a eficácia da mesma já que esta deveria, numa situação ideal, ser independente tanto do assistente utilizado como dos casos de uso para os quais se tirou partido da mesma.

Ainda relativamente à arquitetura, em particular da arquitetura da aplicação smart home, poderia ser explorada a implementação de arquiteturas a tirar partido de *CQRS* e *Event Sourcing*[19] de diferentes formas já que, como vimos anteriormente, o padrão CQRS é flexível para tal. Os serviços para os tipos de dispositivos e funcionalidades presentes nesta arquitetura poderiam também vir a mostrar-se pouco viáveis em aplicações de outros assistentes digitais que não o Google Assistant, pelo que poderia ser feito algum trabalho neste sentido.

Enquanto que o sistema de autenticação está simplesmente a simular a autenticação dos utilizadores, no futuro este teria de ser implementado e partilhado pelas aplicações. Enquanto que a autenticação só é exigida pela aplicação smart home, esta seria também necessária na aplicação conversacional para fazer o controlo da informação do utilizador, como o seu e-mail e nome, de modo a melhorar a experiência com o mesmo. Assim, seria necessário o desenvolvimento de uma aplicação adicional, responsável pela autenticação dos utilizadores no sistema.

A solução final conseguida permite os utilizadores tirarem partido da aplicação smart home, através de um *middleware*, a aplicação conversacional. A realidade é que, ao utilizar a aplicação conversacional para tirar partido das funcionalidades acrescidas que foram implementadas, estão a ser perdidas algumas das funcionalidades existentes no assistente, que estariam disponíveis através da utilização direta da aplicação smart home. Um exemplo disto é a distinção dos dispositivos nas diferentes divisões da casa do utilizador, esta funcionalidade teria de ser implementada na aplicação conversacional. A utilização da

1 https://www.rabbitmq.com/
2 https://kafka.apache.org/

aplicação conversacional para interação com os dispositivos só se justifica para situações muito particulares, pelo que, para as restantes interações será mais vantajosa a utilização direta da aplicação smart home e, portanto, do módulo NLP que a Google oferece.

9.3 CONSIDERAÇÕES FINAIS

Por fim, o sistema desenvolvido cumpre com os objetivos inicialmente propostos, alinhado com uma arquitetura cujo o foco está na escalabilidade, requisito cada vez mais comum em aplicações deste contexto. Foi possível adquirir conhecimento não só ao nível de desenho de arquiteturas como também de algumas tecnologias recentes, sendo grande parte destas dentro do domínio de desenvolvimento na *cloud*. Sendo o desenvolvimento de aplicações para o Google Assistant uma prática recente, as constantes alterações na documentação foram um desafio, já que foi necessário estar constantemente preparado para fazer as mudanças necessárias nas aplicações, em especial na aplicação smart home, cuja documentação era alterada a cada 3 meses, aproximadamente. Estas alterações permitiram também estar em contacto com elementos das equipas de desenvolvimento do Google Assitant para esclarecer algumas dúvidas que foram surgindo e até pedir sugestões de abordagens para os diversos desafios encontrados, tornando toda a experiência de desenvolvimento mais enriquecedora.

BIBLIOGRAFIA

[1] Rohan Killedar Abhay Dekate Chaitanya Kulkarni. "Study of Voice Controlled Personal Assistant Device". Em: (2016), pp. 42–46. ISSN: 2231-2803.

[2] Amazon. *Alexa Skills Kit*. 2019. URL: https://developer.amazon.com/docs/alexa-design/get-started.html.

[3] George Anders. *Alexa, Understand Me*. 2017. URL: https://www.technologyreview.com/s/608571/alexa-understand-me/.

[4] Apple. *Siri for Developers*. 2019. URL: https://developer.apple.com/sirikit/.

[5] Nil Goksel Aras Bozkurt. "Technology Renovates itself: Key concepts on Intelligent Personal Assistants (IPAs)". Em: 2018, pp. 4291–4297. DOI: 10.21125/edulearn.2018.1082.

[6] Oktay Bahceci. "Analysis and Comparison of Intelligent Personal Assistants". Em: (2016).

[7] Oktay Bahceci. "Understanding adoption of intelligent personal assistants: A parasocial relationship perspective". Em: (2018).

[8] P. Bellavista e A. Zanni. "Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP". Em: *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. Set. de 2016, pp. 1–6. DOI: 10.1109/RTSI.2016.7740614.

[9] Kelli Kemery Christi Olson. *Voice report: From answers to action: customer adoption of voice technology and digital assistants*. 2019. URL: https://advertiseonbing-blob.azureedge.net/blob/bingads/media/insight/whitepapers/2019/04\%20apr/voice-report/bingads_2019voicereport.pdf.

[10] Evans. "Domain-driven design - tackling complexity in the heart of software". Em: 2004.

[11] Martin Fowler. *CQRS*. 2019. URL: https://martinfowler.com/bliki/CQRS.html.

[12] David Garlan e Mary Shaw. "An Introduction to Software Architecture". Em: jan. de 1993, pp. 1–39.

[13] Google. *Cloud Firestore documentation*. 2019. URL: https://cloud.google.com/firestore/docs/.

[14] Google. *Dialogflow documentation*. 2019. URL: https://cloud.google.com/dialogflow/docs/.

[15] Google. *Google Assistant Documentation*. 2019. URL: https://developers.google.com/actions/.

[16] Google. *Google Cloud Functions documentation*. 2019. URL: https://cloud.google.com/functions/docs/.

[17] Google. *Google Design guidelines*. 2019. URL: https://designguidelines.withgoogle.com.

[18] Joseph E. Gonzalez Johann Schleier-Smith Vikram Sreekanti Alexey Tumanov Joseph M. Hellerstein Jose Faleiro e Chenggang Wu. "Serverless Computing: One Step Forward, Two Steps Back". Em: (2018).

[19] Jaap Kabbedijk, Slinger Jansen e Sjaak Brinkkemper. "A case study of the variability consequences of the CQRS pattern in online business software". Em: (jul. de 2012). DOI: 10.1145/2602928.2603078.

[20] Khawir Mahmood, Tauseef Rana e Abdur Rehman Raza. "Singular Adaptive Multi-Role Intelligent Personal Assistant (SAM-IPA) for Human Computer Interaction". Em: dez. de 2018, pp. 35–41. DOI: 10.1109/ICOSST.2018.8632189.

[21] Mirjana Maksimovic et al. "Raspberry Pi as Internet of Things hardware: Performances and Constraints". Em: jun. de 2014.

[22] Bertrand Meyer. *Object-Oriented Software Construction*. 1st. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. ISBN: 0136290493.

[23] Microsoft. *Cortana Skills Kit*. 2019. URL: https://docs.microsoft.com/en-us/cortana/skills/overview.

[24] Microsoft. *Design a DDD-oriented microservice*. 2019. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice.

[25] V. Patchava, H. B. Kandala e P. R. Babu. "A Smart Home Automation technique with Raspberry Pi using IoT". Em: *2015 International Conference on Smart Sensors and Systems (IC-SSS)*. Dez. de 2015, pp. 1–4. DOI: 10.1109/SMARTSENS.2015.7873584.

[26] Thinagaran Perumal, Soumya Kanti Datta e Christian Bonnet. "IoT device management framework for smart home scenarios". Em: out. de 2015, pp. 54–55. DOI: 10.1109/GCCE.2015.7398711.

[27] Arsénio Reis et al. "Using Intelligent Personal Assistants to Strengthen the Elderlies' Social Bonds : A preliminary evaluation of Amazon Alexa, Google Assistant, Microsoft Cortana, and Apple Siri". Em: mai. de 2017, pp. 593–602. ISBN: 978-3-319-58699-1. DOI: 10.1007/978-3-319-58700-4_48.

[28] Mike Roberts. *Serverless Architectures*. 2019. URL: https://martinfowler.com/articles/serverless.html.

[29] Roland Steinegger et al. "Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications". Em: abr. de 2017.

[30] Amrita Tulshan e Sudhir Namdeorao Dhage. "Survey on Virtual Assistant: Google Assistant, Siri, Cortana, Alexa: 4th International Symposium SIRS 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers". Em: jan. de 2019, pp. 190–201. ISBN: 978-981-13-5757-2. DOI: 10.1007/978-981-13-5758-9_17.

[31] K Venkatesh et al. "IoT Based Home Automation Using Raspberry Pi". Em: *Journal of Advanced Research in Dynamical and Control Systems* 10 (jul. de 2018), pp. 1721–1728.