Universidade do Minho
Escola de Engenharia

Diogo Miguel das Neves Francisco

Tecnologias Bluetooth para Ambient Assisted Living (AAL)

Tese de Mestrado
Mestrado Integrado em Engenharia de Comunicações

Trabalho efetuado sob a orientação de
Professor Doutor Jorge Miguel Nunes dos Santos Cabral

Outubro de 2014

Agradecimentos

Em primeiro lugar, sem qualquer dúvida, agradecer à minha mãe, Ana Paula Coelho, por todos os sacrifícios feitos ao longo destes passados 24 anos. Sem dúvida alguma, não estaria onde estou agora se não tivesse tido todo o apoio e suporte que tive e a liberdade para escolher o meu caminho.

Ao resto da minha família, que sempre me apoiou quando assim foi necessário.

Ao Professor Jorge Cabral, pelo convite que me estendeu e pela oportunidade de aprender. Reconheço também a confiança que depositou no meu trabalho.

À Daniela Filipa Silva, que ao longo destes anos foi um dos meus grandes pilares, onde sempre pude encontrar companheirismo, amizade, paciência e apoio, quando assim precisei.

Agradecer também aos meus amigos, Diogo Ferreira, Nelson Pinto, João Brito, David Cunha, Márcia Ferreira, Rita Barbosa e Sara Barros com os quais passei muitos e bons momentos. Foram, sem dúvida, pessoas que me ajudaram ao longo deste meu percurso, quer pela companhia em momentos mais relaxados, quer pelas horas de estudo e trabalho que partilhamos.

O meu agradecimento final segue para o Grupo de Sistemas Embebidos da Universidade do Minho que me deu a oportunidade de trabalhar inserido num grupo fantástico de pessoas. É extremamente gratificante olhar para trás e perceber o quanto crescemos todos ao longo deste último ano. Fica um agradecimento especial ao Jorge Miranda pela experiência que me transmitiu e pelo excelente trabalho que também desenvolveu no projeto *CareStore*.

Resumo

A população europeia está a envelhecer e as previsões apontam para que esta seja uma tendência que se irá manter nas próximas décadas. Isto levanta desafios de ordem económica e social para os quais a sociedade atual não está preparada para dar resposta.

Com o objetivo de evitar uma redução drástica no nível de qualidade de vida, à medida que as pessoas envelhecem, surgem as plataformas de Ambient Assisted Living. Estas plataformas aparecem no sentido de modernizar e baixar o custo dos sistemas de saúde que garantem o acompanhamento médico das pessoas idosas, passando esse acompanhamento a poder ser feito no conforto dos lares ou na sua própria habitação, de uma forma fácil.

O projeto *CareStore* implementa uma plataforma *plug & play* de Ambient Assisted Living que permite a monitorização da saúde dos seus utilizadores. Esta plataforma é baseada em ferramentas *open source* e permite a interoperabilidade entre dispositivos médicos de diversos fabricantes. De modo a garantir a flexibilidade da plataforma e que esta é adequada ás necessidades específicas de cada local onde é instalada, esta disponibiliza um *marketplace* de onde podem ser descarregadas as aplicações de saúde mais adequadas. Isto permite que os cidadãos ou profissionais de saúde possam transportar os seus equipamentos de saúde e instalá-los facilmente no local do utilizador, com um custo nulo ou reduzido. A plataforma *CareStore* foi pensada de modo a ser facilmente implementada, não requerendo a intervenção de técnicos especializados

Neste documento, serão discutidos os passos de desenvolvimento do *software* relacionado com as funcionalidades Bluetooth da Common Recognition Interface Platform, um dos subsistemas do *CareStore*, desenvolvido na Universidade do Minho pelo Grupo de Investigação de Sistemas Embebidos e responsável pela identificação de utilizadores e pela recolha de dados médicos de dispositivos especializados.

Abstract

The european population is aging and the forecasts say that this is a steady tendency for the following decades. This raises economic and social challenges that the society, as it is today, doesn't have an answer for.

With the objective of avoiding a drastic reduction of quality of life in the ageing population, Ambient Assisted Living platforms are emerging. These platforms arise with the objective of modernizing and lowering the costs of healthcare systems that ensure the medical supervision of the elderly, making it so that that supervision can be made in the comfort of nursing homes or their households, in a very simple manner.

The CareStore project implements an Ambient Assisted Living plug & play platform, that allows the health monitoring of its users. This platform is based on open source tools and allows the interoperability of medical devices from different manufacturers. As a way to guarantee that this platform is flexible and adequate to the specific requirements of each deployment place, it implements a marketplace from which the users can download health applications, acording to their needs. This allows the citizens and the health caregivers to transport their personal health devices and easily install them on the user premises, with zero or low cost. The CareStore platform was designed to be easily installed, not requiring the intervention of specialized staff.

In this document, the steps to the development of the software related to the Bluetooth functionality of the Common Recognition Interface Platform, one of the subsystems of the CareStore project, developed at Minho University by the Embedded Systems Research Group, will be discussed. This subsystem is responsible for the identification of the platform users and for gathering medical data from the specialized devices.

Conteúdo

Acronyms

AAL Ambient Assisted Living.

ACL Asynchronous Connection-Less.

AMP Alternative MAC/PHY.

API Application Program Interface.

BLE Bluetooth Low Energy.

BLP Blood Pressure Profile.

CAALHP Common Ambient Assisted Living Home-Platform.

CPP Cycling Power Profile.

CRIP Common Recognition Interface Platform.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

EIR Extended Inquiry Response.

FHSS Frequency Hopping Spread Spectrum.

GLP Glucose Profile.

HCI Host Controller Interface.

HDP Health Device Profile.

HFP Hands-Free Profile.

HRP Heart Rate Profile.

HTP Health Thermometer Profile.

HTTP HyperText Transfer Protocol.

IEEE Institute of Electrical and Electronics Engineers.

IPC Inter-Process Communication.

ISM Industrial, Scientifical and Medical.

L2CAP Logical link control and adaptation protocol.

MAC Medium Access Control.

MCAP Multi-Channel Adaptation Protocol.

MITM Man In The Middle.

NDEF NFC Data Exchange Format.

NFC Near Field Communication.

OCF Opcode Command Field.

OGF Opcode Group Field.

OOB Out Of Band.

OOP Object Oriented Programming.

PIN Personal Identification Number.

RPC Remote Procedure Call.

RSCP Running Speed and Cadence Profile.

RSSI Received Signal Strength Indication.

SAP SIM Access Profile.

SDP Service Discovery Protocol.

SIM Subscriber Identity Module.

SSP Secure Simple Pairing.

UART Universal Asynchronous Receiver/Transmitter.

URI Uniform Resource Identifier.

USB Universal Serial Bus.

WSP Weight Scale Profile.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Naturalmente, o sistema que foi desenvolvido no contexto desta dissertação deve ser explicado, não só de um ponto de vista tecnológico, mas também de um ponto de vista social, de modo a que se perceba a utilidade da sua criação e quais as suas características.

Neste capítulo, serão abordados alguns temas que permitem contextualizar e melhor compreender o trabalho desenvolvido para o projeto em questão. Nomeadamente:

- Necessidade, cada vez mais evidente, de sistemas para AAL e em que contextos se torna apropriada a implementação de sistemas com esta finalidade;
- O que é, quais os objetivos e quais as características que tornam o projeto *CareStore* único;
- O uso de tecnologias Bluetooth no projeto *CareStore*.

1.1 Ambient Assisted Living

Segundo algumas previsões, feitas por órgãos ligados à União Europeia, nas próximas décadas iremos deparar-nos com uma série de novos desafios, relacionados com alterações demográficas sérias na população europeia, para os quais a nossa sociedade ainda não possui mecanismos de resposta eficientes. De acordo com um estudo demográfico, realizado pela União Europeia [18]:

- Aumento substancial dos níveis absolutos e percentuais da população idosa(65+ anos). Entre 2010 e 2060 deverá haver um aumento de 87.5 para 152.6 milhões. Em termos percentuais, isto representa um aumento de 17% para 30%;

- Aumento significativo dos custos percentuais em saúde e cuidados médicos, devido às necessidades acrescidas da população sénior;

- Diminuição percentual da população trabalhadora a longo prazo, devido ao envelhecimento da população. Segundo as previsões, após um crescimento de 1,5% até 2020, poder-se-á assistir a uma reversão nessa tendência daí para frente, verificando-se um decréscimo de 11,75% até 2060.

- Crescimento da esperança média de vida. Prevê-se que um aumento médio de 7.2 anos de vida entre 2008 e 2060. Teremos portanto uma população cada vez mais envelhecida, o que pressupõe serem necessários ainda mais cuidados;

- Índices de fertilidade bastante mais baixos no século XXI, comparativamente às décadas de 60 e 70. Devido ao *boom* de fertilidade pós 2ª Guerra Mundial, era possível observar índices de 2,5 crianças por mulher, enquanto que atualmente, se podem observar valores a rondar as 1,6 crianças. No entanto, tem-se assistido a um aumento de fertilidade desde o início do século XXI, o que pode significar uma reversão desta tendência.

Embora não seja possível evitar que as pessoas envelheçam, é possível acompanhar, melhorar e facilitar a sua experiência de envelhecimento. É nesse sentido que surgem as plataformas de *Ambient Assisted Living*.

Disponibilizada pelo Ministério Alemão de Educação e Investigação[1], esta é uma das definições que mais se enquadra com aquilo que é o AAL:

> *"Ambient Assisted Living"(AAL) are concepts, products and services which combine new technologies and the social environment in order to improve quality of life in all periods of life."*

O AAL não é mais que um conceito suportado por plataformas e mecanismos usados com os objetivos [19] de:

[1] `http://www.bmbf.de/en/`

- Maximizar o tempo útil de vida das pessoas, aumentando a autonomia, mobilidade e confiança das mesmas;
- Promover um estilo de vida mais saudável para indivíduos em risco;
- Apoiar as famílias, prestadores de cuidados e organizações de saúde;
- Aumentar a segurança, prevenir o isolamento e manter um rede funcional de contactos à volta do indivíduo.

De certa maneira, o conceito de AAL não está diretamente relacionado com a população idosa, na medida em que não é necessário ser idoso para perceber os benefícios que advêm do uso de uma plataforma com este fim. No entanto, é também um facto adquirido que à medida que o indivíduo envelhece se vai perdendo qualidade de vida, tornando-se mais dependente e vulnerável. Tendo como base as estatísticas apresentadas acima, torna-se cada vez mais evidente a necessidade de sistemas que cumpram estes objetivos.

Para terminar esta secção, é disponibilizada a imagem 1.1, que apresenta alguns dos elementos vulgarmente presentes num sistema de suporte ao AAL.

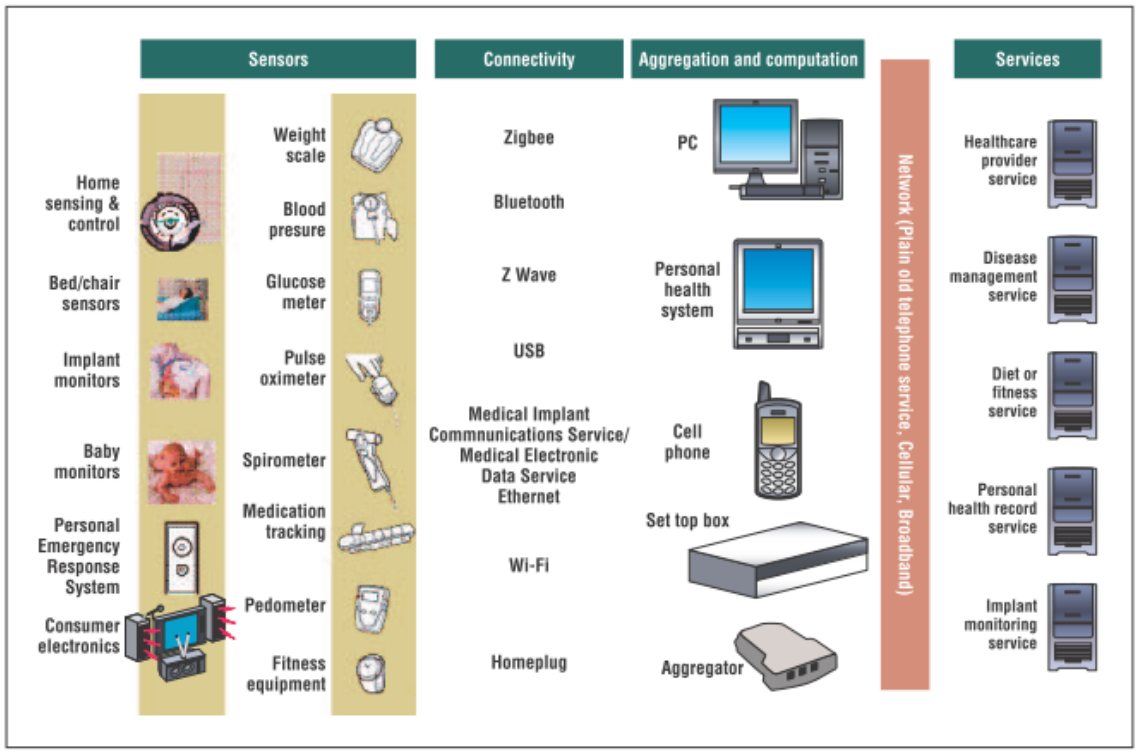


Figura 1.1: Elementos utilizados no Ambient Assisted Living (imagem retirada de [1]).

Por análise à figura 1.1, podemos verificar que um sistema de AAL comum é composto por sensores e dispositivos de recolha, processamento e apresentação de informação. De modo a interligar esses dispositivos, existem várias tecnologias de comunicação. O sistema formado por estes elementos pode disponibilizar serviços acessíveis através da *Internet* de modo a garantir aos seus utilizadores um método de obter a informação recolhida pelo sistema.

Em termos de sensores, estes são o que seria de esperar num sistema de AAL. É possível ter balanças, sensores de pressão arterial, sensores de pulsação cardíaca ou até implantes corporais. Geralmente, os dados recolhidos por este tipo de dispositivos são comunicados de modo a poderem ser recolhidos, processados e apresentados num computador, *smartphone* ou mesmo um sistema personalizado para esse efeito.

Relativamente às tecnologias envolvidas no processo de comunicação dos dados médicos, as mais comuns são Bluetooth, ZigBee ou mesmo WiFi, no entanto é possível a utilização de bastantes mais tecnologias que as apresentadas na figura em questão.

1.2 CareStore

O projeto *CareStore*[20] visa a criação de uma plataforma *open source* de *Ambient Assisted Living*(AAL), que permita a fácil implementação de aplicações e dispositivos médicos, sem ser necessária a intervenção de técnicos especializados.

Trata-se de um projeto co-financiado pela União Europeia, por via do programa *Seventh Framework*, sendo o mesmo coordenado pelo *CareStore Consortium*, composto por 3 empresas(Sekoia ApS, Romex, Rommerskirchen GbR) e 3 Universidades(Aarhus University, Universidade do Minho, Institut de Recherche en Systèmes Electroniques Embarqués) empenhadas em fazer do *AAL* uma realidade presente no dia-a-dia das populações idosas da União Europeia.

À Universidade do Minho foi atribuída a tarefa de desenvolver e testar um dos subsistemas da plataforma *CareStore*, cuja arquitetura será apresentada numa secção mais apropriada.

Naturalmente, já existem alguns sistemas semelhantes ao *CareStore* no mercado, mas devido às razões explicadas anteriormente, ainda há muito trabalho a ser feito nesta área e a margem de progressão é enorme tendo em conta que apenas recentemente se começou a olhar para esta área com mais alguma preocupação. O *CareStore* insere-se na tentativa de fazer do AAL uma realidade e trazer todos os benefícios já referidos anteriormente às populações necessitadas.

Tendo isto, os objetivos do projeto *CareStore* são os seguintes [8] [21]:

- Desenvolvimento de uma plataforma *open source* que vá de encontro aos objetivos do AAL;
- Interação com a plataforma deve ser o mais transparente possível;
- Criação de um mercado de aplicações que forneça os *drivers* necessários para diferentes dispositivos médicos e aplicações de saúde com diferentes objetivos;
- Assegurar a segurança da informação sensível presente no sistema e garantir o acesso ao sistema apenas a agentes e dispositivos autorizados;
- Integração do sistema domiciliário de recolha de dados médicos com dispositivos de múltiplos fabricantes;

- Implementação de mecanismos que garantam a interoperabilidade da plataforma;

- Garantir uma redução de custos temporais e monetários ao facilitar o acesso aos dados médicos dos utilizadores do sistema.

Os objetivos apresentados são bastante vagos e terão de ser explorados em secções mais pertinentes desta dissertação, onde será explicado, com bastante detalhe, o que foi feito para cumprir os objetivos referidos e como foi feito. Algo que também já foi mencionado é o facto de não caber à Universidade do Minho encontrar a solução para todos estes objetivos, visto este ser um trabalho conjunto entre várias entidades. Ou seja, os detalhes técnicos de implementação de alguns dos módulos desenvolvidos serão, com toda a certeza, melhor explicados pelas entidades responsáveis pelo desenvolvimento desses mesmo módulos.

Tal como já foi referido, alguns sistemas no mercado (apresentar-se-ão vários exemplos no próximo capitulo) já cumprem alguns destes objetivos. No entanto, o que se nota é que são sistemas fechados, com *Software* e *Hardware* proprietários e que, para além disto, são pouco personalizáveis, complexos e nada intuitivos de usar. Tendo em conta que já é possível adquirir, com facilidade, dispositivos para recolha de dados médicos, e que bastantes desses dispositivos permitem exportar os dados recolhidos, começa a ser importante pensar em questões de interoperabilidade. Alguns sistemas surgiram numa altura em que a tecnologia ainda não permitia que a interoperabilidade fosse possível e este não era ainda um objetivo crucial. Atualmente, é fundamental garantir a interoperabilidade e o projeto *CareStore* implementa e aproveita alguns mecanismos que garantem que a plataforma se mantém em funcionamento com uma ampla gama de dispositivos.

Para o projeto *CareStore*, a facilidade em usar o sistema é algo que é essencial e que se torna evidente, se pensarmos naqueles que vão ser os seus principais utilizadores. É importante que a interação com o sistema apenas exista quando necessário e que seja rápida e simples. A própria instalação do sistema segue uma filosofia *Plug & Play*, de modo a garantir que o sistema é rapidamente posto em funcionamento, com pouquíssima configuração necessária.

1.3 Bluetooth aplicado ao AAL

Os objetivos do AAL podem ser concretizados com recurso a várias tecnologias, tal como visto na imagem 1.1. Uma das tecnologias mais proeminentes neste campo é o Bluetooth[2].

O Bluetooth trata-se de uma tecnologia de comunicação sem fios de curto alcance, que apresenta algumas características que a tornam especialmente útil no contexto do AAL:

- Curto alcance;
- Mecanismos de resistência a interferências eletromagnéticas;
- Baixo consumo energético, principalmente no modo *Low Energy*;
- Existência de especificações de perfis adaptados à comunicação com dispositivos médicos;
- Simplicidade de uso.

Para além disto, apesar de este tema não ter sido explorado a fundo no projeto *CareStore*, o *standard* Bluetooth 4.0 específica um modo de operação *Low Energy*. Esta tecnologia apresenta consumos bastante inferiores ao funcionamento clássico da tecnologia [22] [23]. O Bluetooth *Low Energy* apenas agora começa a ser explorado completamente, com o lançamento de dispositivos com *software*capaz de dar suporte ao seu funcionamento.

Num sistema de AAL, esta tecnologia é geralmente utilizada com o objetivo de garantir a conetividade entre a plataforma de recolha de dados médicos e os dispositivos médicos propriamente ditos. A quantidade de dispositivos médicos certificados pelo Bluetooth Special Interest Group é bastante considerável[3], o que demonstra que esta é uma tecnologia de eleição no campo dos dispositivos médicos. Isto acontece pois, tal como já foi referido, a especificação Bluetooth já cobre perfis de comunicação[4] especialmente adaptados à troca de dados com dispositivos médicos:

[2] `http://www.bluetooth.com/`
[3] `http://www.bluetooth.com/Pages/Product-Listing.aspx?ProductCategory=16`
[4] `https://www.bluetooth.org/en-us/specification/adopted-specifications`

- Bluetooth *Classic*
  - *Health Device Profile.*

- Bluetooth *Low Energy*
  - *Health Thermometer Profile*;
  - *Heart Rate Profile*;
  - *Blood Pressure Profile*;
  - *Cycling Power Profile*;
  - *Glucose Profile*;
  - *Running Speed and Cadence Profile*;
  - *Weight Scale Profile.*

Ou seja, o Bluetooth *Classic* implementa um perfil apenas ao nível do processo de comunicação Bluetooth propriamente dito, deixando a camada aplicacional em aberto. Ou seja, apenas é definido como são estabelecidas as conexões com os dispositivos médicos e não os dados que estes devem trocar a nível aplicacional. No caso do *Low Energy*, a estratégia é diferente, sendo definido também o modelo de dados que os dispositivos médicos devem utilizar. Devido à quantidade considerável de dispositivos médicos diferentes, foi necessário definir um modelo de dados para cada tipo de dispositivo diferente.

Como já foi referido, no trabalho inserido nesta dissertação, apenas foi abordada a tecnologia Bluetooth *Classic*, visto ser uma tecnologia mais madura e amplamente utilizada. Para além disto, a quantidade de dispositivos que utilizam os perfis *Low Energy*, apenas agora começam a ver a luz do dia.

Capítulo 2

Estado da Arte

Antes de explicar o trabalho desenvolvido no contexto do projeto *CareStore*, é importante introduzir algumas das tecnologias que são utilizadas no *CareStore*e qual o seu estado atual de desenvolvimento, para que seja mais simples a compreensão da parte técnica que será apresentada em capítulos seguintes. Esta análise poderá também dar a entender algumas das decisões de projeto que foram tomadas ao longo do tempo.

Para além disto, é interessante perceber quais os sistemas que tentam ir de encontro aos objetivos do AAL e analisar as suas características, de modo a melhor compreender o que o projeto *CareStore* traz de novo a esta área e quais as vantagens e desvantagens destes outros sistemas.

2.1 Tecnologias do CareStore

2.1.1 Bluetooth

O Bluetooth é uma tecnologia de comunicação sem fios e de curto alcance (geralmente não excede os 100 metros). As características que tornam esta tecnologia tão apelativa são a sua robustez, baixo consumo e baixo custo.

A partir da especificação 4.0 da tecnologia, existem dois tipos de sistema: sistemas Bluetooth *Classic*, também conhecido como *Basic Rate* e sistemas Bluetooth *Low Energy*. Ambos os sistemas incluem mecanismos semelhantes como, por exemplo, descoberta de dispositivos e o estabelecimento de conexões entre dispositivos.

O Bluetooth *Classic* é o tipo de tecnologia Bluetooth mais antigo, tendo sofrido já bastantes modificações ao longo do tempo. No modo *Basic Rate*, apresenta taxas de transmissão (a nível de dados de aplicação) na ordem dos 721.2 kbps, enquanto que no modo *Enhanced Data Rate* pode atingir os 2.1 Mbps. Também é possível um modo de operação de taxa elevada, que atinge os 24 Mbps recorrendo, no entanto, a tecnologias 802.11 que nada tem a ver com Bluetooth.

Relativamente ao Bluetooth *Low Energy*, este tipo de tecnologia apenas é descrito a partir da versão 4.0 da especificação Bluetooth e foi criado a pensar na necessidade de sistemas com menor consumo, custo e complexidade que os sistemas Bluetooth *Classic*. Isto faz com que a taxa de transmissão dos sistemas *Low Energy* seja mais reduzida, atingindo apenas os 0.27 Mbps.

É possível a implementação dos dois sistemas, o que garante que o dispositivo que os implementa possui a capacidade de falar com dispositivos que implementem qualquer um dos sistemas ou ambos.

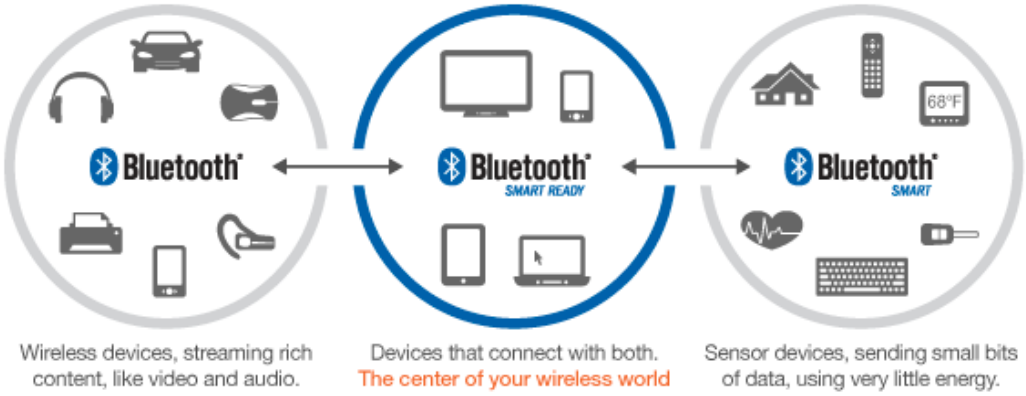


Figura 2.1: Tipos de tecnologia Bluetooth (imagem retirada de [2]).

Como é possível observar na figura 2.1, os dispositivos Bluetooth podem ser distribuídos por três categorias:

- Bluetooth
  - Este tipo de dispositivo implementa o Bluetooth *Classic* e é utilizado em aplicações que possuam menos restrições energéticas e que necessitam de

uma taxa de transmissão mais elevada. Este tipo de dispositivos podem interagir com dispositivos Bluetooth ou Bluetooth *Smart Ready*, no entanto, não estão habilitados a interagir com dispositivos Bluetooth *Smart*.

- Bluetooth *Smart*
  - Este tipo de dispositivo implementa o Bluetooth *Low Energy* e é utilizado em aplicações que possuam mais restrições energéticas e onde a taxa de transmissão não seja tão relevante. Este tipo de dispositivos podem interagir com dispositivos Bluetooth *Smart* ou Bluetooth *Smart Ready*, estando desabilitados a interagir com dispositivos que implementem Bluetooth *Classic*.
- Bluetooth *Smart Ready*
  - É com este tipo de dispositivos que é feita a ponte entre dispositivos Bluetooth e Bluetooth *Smart*. Este tipo de dispositivo implementam quer Bluetooth *Classic*, quer Bluetooth *Low Energy* podendo interagir com todos os dispositivos do universo Bluetooth.

Um sistema Bluetooth é composto por um e um só *Host* e um ou mais *Controllers*.

O *Host* não é mais que uma entidade lógica que implementa as camadas mais altas da pilha Bluetooth. É importante referir que as tecnologias *Classic* e *Low Energy* possuem uma pilha um pouco distinta, partilhando apenas algumas camadas da pilha Bluetooth.

Um *Controller* é também uma entidade lógica que implementa as camadas mais baixas da pilha Bluetooth, associadas aos processos físicos de comunicação da pilha Bluetooth. Naturalmente, um *Controller* que implemente o Bluetooth *Classic* é diferente de um *Controller Low Energy*.

Estes dois módulos são interligados por uma camada denominada *Host Controller Interface* (HCI), que é melhor explicada na secção 4.1. A importância desta camada é a garantia que dá relativamente à independência entre o *software* e o *hardware*.

A figura 2.2 demonstra as combinações *Host*/*Controller* que é possível encontrar num sistema Bluetooth.

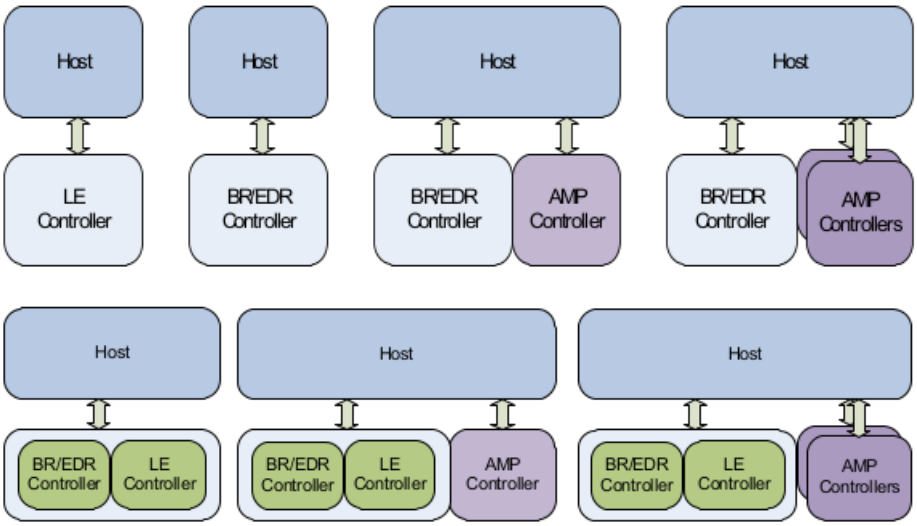


Figura 2.2: Combinações *Host*/*Controller* de um sistema Bluetooth (imagem retirada de [3]).

Feita esta pequena introdução às tecnologias Bluetooth, é interessante ter uma visão geral da pilha Bluetooth e quais as camadas que a compõem. Esta é apresentada na figura 2.3.

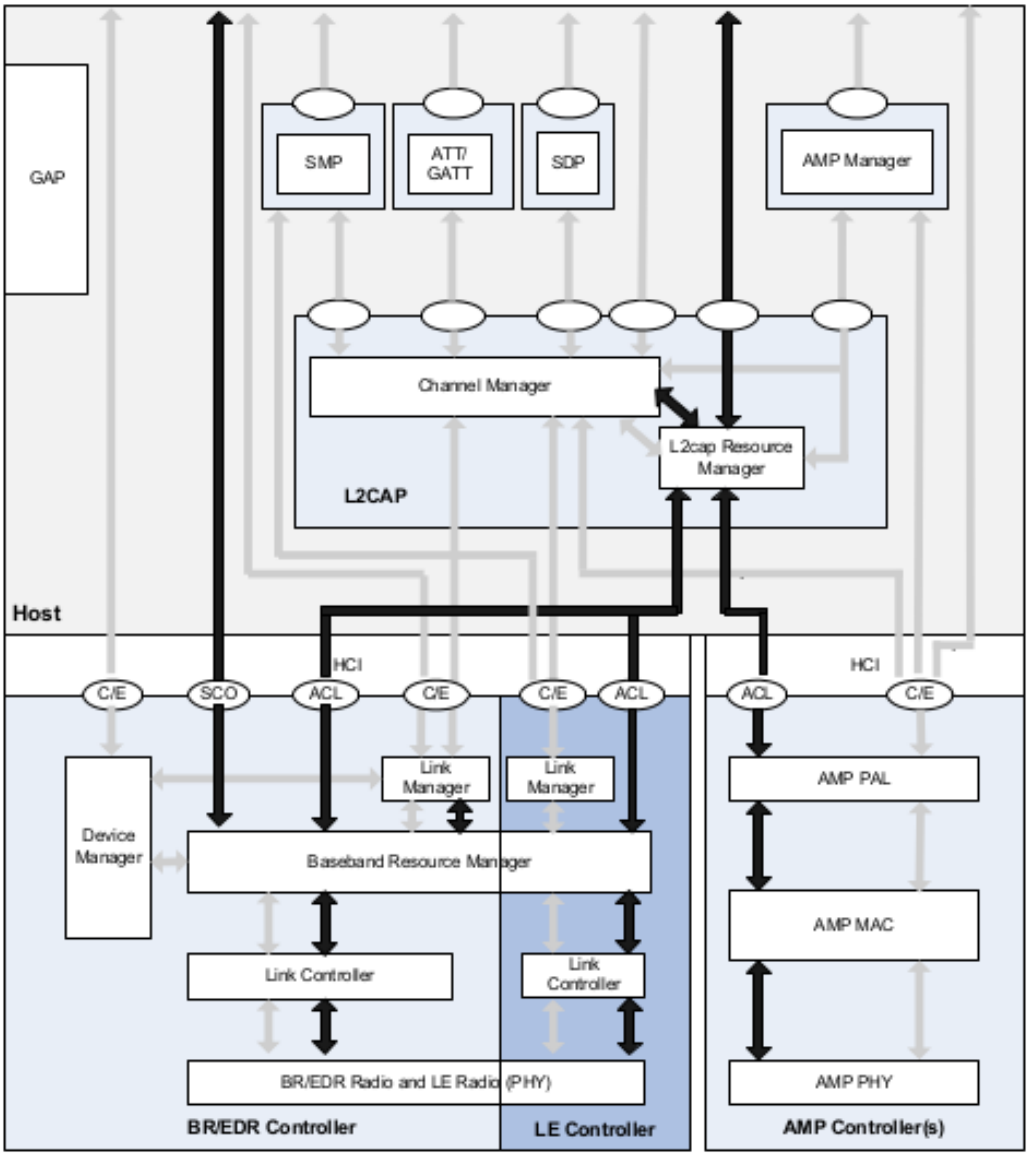


Figura 2.3: *Overview* da pilha Bluetooth (imagem retirada de [3]).

A partir deste ponto, apenas será descrita a componente Bluetooth *Classic* da especificação Bluetooth, tendo em conta que o trabalho descrito neste documento se foca nesse tipo de tecnologias.

2.1.1.1 Bluetooth Classic

O Bluetooth *Classic* opera nas bandas de frequência ISM, mais especificamente a de 2.4 GHz. Utiliza 79 canais, sendo que cada um tem uma largura de banda de 1 MHz e está separado do canal adjacente por 1 MHz. Como já foi referido, utilizando o modo *Enhanced Data Rate*, isto permite taxas de transmissão até 2.1 Mbps.

De modo a reduzir a interferência resultante do uso destas frequências, que já se encontram ocupadas por outras tecnologias de comunicação sem fios (a mais relevante sendo a tecnologia WiFi), é implementado um esquema de *Frequency Hopping Spread Spectrum* (FHSS), que permite que o canal usado durante um processo de comunicação Bluetooth seja alterado entre a receção ou transmissão de pacotes, fazendo com que o efeito de possíveis interferências seja minimizado. Estes saltos entre frequências ocorrem 1600 vezes por segundo. Se alguns canais estiverem demasiado congestionados, o mecanismo de *Frequency Hopping* não permite a utilização desses canais.

Num processo típico de comunicação entre dispositivos Bluetooth, o canal físico que é utilizado é partilhado uma série de dispositivos sincronizados e que seguem o mesmo padrão de saltos de frequência. Ao dispositivo que fornece os parâmetros de sincronização é dado o nome de *Master*. Todos os outros dispositivos que se sincronizam com o relógio e padrão de saltos de frequência do *Master*, é dada a designação de *Slaves*. No Bluetooth *Classic*, existem no máximo 7 *Slaves*.

A rede formada por via da conexão entre o *Master* e os *Slaves* é designada de *Piconet*. Numa *Piconet* pode haver um e um só *Master* e não existe comunicação direta entre *Slaves*. Um *Slave* numa *Piconet* pode ser o *Master* noutra *Piconet* e vice-versa. Quando duas *Piconets* partilham elementos, é formada uma *Scatternet*, se bem que este conceito não é adotado na especificação Bluetooth.

A figura 2.4 ilustra os conceitos apresentados.

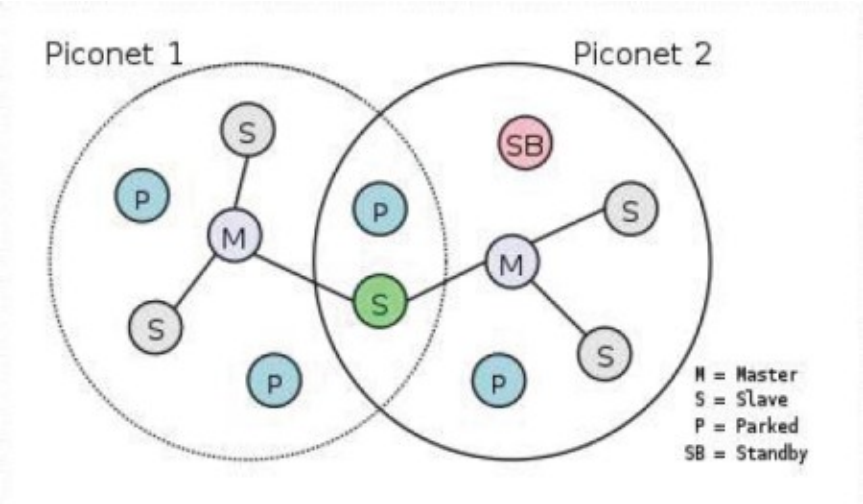


Figura 2.4: Estrutura de redes Bluetooth (imagem retirada de [3]).

A identificação de dispositivos Bluetooth é feita por via de um endereço MAC (Medium Access Control) de 48 bits (6 bytes), onde os primeiros 3 *bytes* identificam a organização que fabrica o dispositivo Bluetooth e os restantes 3 *bytes* são de livre definição por essa mesma organização.

Para terminar, é importante introduzir algumas das operações típicas num processo de comunicação Bluetooth como a descoberta e anúncio de dispositivos e o emparelhamento de dispositivos.

Começando pelo processo de descoberta de dispositivos ou, como é designado na especificação Bluetooth, *Inquiry Procedure*, este processo permite a um dispositivo Bluetooth descobrir outros dispositivos que estejam configurados como estando visíveis. Ou seja, é possível configurar um dispositivo de modo a que este não responda a pedidos de *Inquiry*, o que faz com que este esteja invisível do ponto de vista do dispositivo que faz a descoberta de dispositivos.

Em termos dos dados que são recolhidos num processo de descoberta de dispositivos Bluetooth, até à versão 2.1 da tecnologia Bluetooth, a única informação recolhida dos dispositivos encontrados era o endereço do dispositivo. Com o desenvolvimento da tecnologia, foi-se tornando evidente que o processo de descoberta poderia ser utilizado para que os dispositivos visíveis fornecessem mais informação que apenas o endereço, de modo a permitir ao dispositivo que despoleta o processo de *Inquiry* ter acesso a informação que permita perceber melhor que tipo de dispositivo é que dete-

tou e quais os serviços que disponibiliza.

Para ultrapassar esta limitação, a partir da versão 2.1 da especificação Bluetooth surgiu o *Extended Inquiry Response* que permite que um dispositivo anuncie não só o seu endereço mas bastante mais informação:

- Endereço do dispositivo;
- Nome do dispositivo;
- Classe do dispositivo;
- Lista de serviços disponibilizados;
- Potência de transmissão.

Tendo acesso à classe de dispositivo e lista de serviços que o dispositivo descoberto disponibiliza, é possível ao dispositivo que efetua a descoberta decidir se o dispositivo encontrado implementa funcionalidades do seu interesse.

Relativamente ao emparelhamento de dispositivos, este processo é talvez o mais associado às tecnologias Bluetooth e surge, geralmente, após um processo de descoberta de dispositivos. O processo de emparelhamento basicamente serve para garantir que dois dispositivos se reconhecem mutuamente e autorizam a formar conexões entre si. Feito o emparelhamento entre dois dispositivos, o estabelecimento de conexões entre esses dois dispositivos não exige que seja repetido o processo de emparelhamento. Geralmente, o processo de emparelhamento envolve interação com o utilizador, na medida em que este tem de selecionar o dispositivo que pretende emparelhar e autorizar o emparelhamento, com recurso a alguns mecanismos que serão explicado seguidamente.

Tal como a descoberta de dispositivos, o processo de emparelhamento de dispositivos tem sofrido alterações ao longo do desenvolvimento da especificação Bluetooth.

Até à versão 2.1, existia aquilo a que agora se refere como *Legacy Pairing*. Com este esquema de emparelhamento, o utilizador era obrigado a inserir um PIN (Personal Identification Number) alfanumérico de 16 bits, sendo esse PIN introduzido em ambos os dispositivos. Dispositivos sem capacidades de *input* eram obrigados a

definir um PIN fixo, *hardcoded* na memória do dispositivo. Este esquema de emparelhamento apresenta-se como inseguro devido ao tamanho limitado do PIN e ao facto deste ser o único fator de entropia.

Portanto, a partir da versão 2.1 da especificação Bluetooth, foi criado o *Secure Simple Pairing* (SSP). Este mecanismo de emparelhamento utiliza criptografia de chave pública e não obriga à introdução de PIN's alfanuméricos. O SSP apresenta 4 modelos de associação:

- *Just Works*
  - Este modelo de associação faz com que a interação com o utilizador seja desnecessária, a não ser uma possível confirmação do processo de emparelhamento. Por ter esta característica, é geralmente utilizado quando os dispositivos tem capacidades de IO limitadas. Apesar de não ser necessária a introdução de um PIN, este modelo de associação é mais seguro que o modelo de *Legacy Pairing*.
- *Numeric Comparison*
  - Se ambos os dispositivos possuírem um modo de exibir informação e pelo menos um deles tiver um modo de confirmar a operação de emparelhamento, este é um possível modelo de associação a ser utilizado. Com este modelo, ambos os dispositivos apresentam um código numérico de 6 algarismos, sendo que o utilizador confirma ou cancela a operação de emparelhamento por verificação da igualdade dos códigos. Este mecanismo garante a proteção contra ataques *Man In The Middle* (MITM).
- *Passkey Entry*
  - Este método é semelhante ao anterior, no entanto é utilizado quando um dos dispositivos tem apenas capacidade de saída e o outro apenas tem capacidade de entrada. O dispositivo com capacidades de saída apresenta um código numérico de 6 algarismos, que é introduzido pelo utilizador no dispositivo com capacidade de entrada. Se o código coincidir, os dispositivos são emparelhados. Tal como no caso anterior, a proteção contra ataques MITM é assegurada.

- *Out of Band*
  - Este modelo utiliza um meio de comunicação diferente do Bluetooth para trocar alguns dados necessários ao processo de emparelhamento, tal como será melhor explicado na secção 5.1.3.1. Visto que as informações necessárias ao emparelhamento são obtidas por outra tecnologia, este modelo de associação elimina a necessidade do processo de descoberta. Em termos de processo de emparelhamento propriamente dito, é utilizado um dos três modelos descritos acima.

De modo resumir a informação apresentada, a figura 2.5 apresenta a relação entre o processos de descoberta, conexão e autenticação.

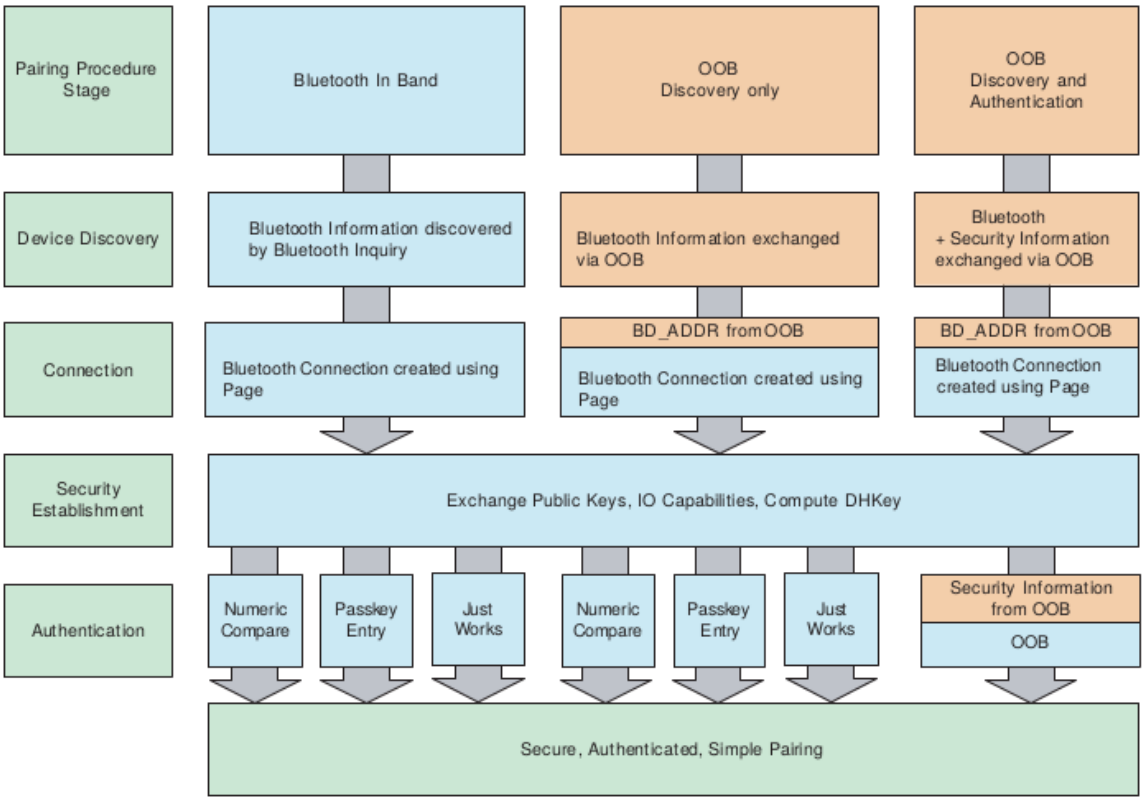


Figura 2.5: Modelos de emparelhamento usando Secure Simple Pairing (imagem retirada de [3]).

Resumidamente, existem dois modos de emparelhar dispositivos: *In Band* e *Out of Band*.

O primeiro é um processo exclusivamente Bluetooth, sendo feita a descoberta de dispositivos, criada uma conexão com o dispositivo que se pretende emparelhar, sendo em seguida tratados os aspetos de segurança da comunicação e utilizado um dos modelos de associação descritos anteriormente.

Relativamente ao processo *Out of Band*, como é possível ver na figura 2.5, este elimina a necessidade de existir o processo de descoberta de dispositivos, sendo que a informação relativa ao dispositivo que se pretende emparelhar é comunicada através de outra tecnologia sem ser Bluetooth. A única diferença que pode existir é a inclusão de processos de autenticação que permitem a troca de informação que torna o processo de emparelhamento OOB mais seguro. Na figura, isso é representado pelo fluxo mais à direita, enquanto que o resto do processo do fluxo do meio é semelhante ao já apresentado para o modo *In Band*.

2.1.2 NFC

Nesta secção será feita uma breve introdução ao NFC, visto que este representa uma componente importante do projeto *CareStore*, sendo utilizado na identificação de utilizadores do sistema e no processo de emparelhamento *Out of Band* de dispositivos Bluetooth, já descrito na secção anterior.

O NFC (Near Field Communication) trata-se de uma tecnologia de comunicação de muito curto alcance (tipicamente menos de 10cm). O NFC opera na banda de frequências ISM de 13.56 MHz e apresenta taxas de transmissão que vão desde 106 kbps até 424 kbps, mediante os modelos de codificação e modulação utilizados.

Um processo de comunicação NFC envolve sempre um *Initiator* e um *Target*. O *Initiator* trata-se geralmente de um dispositivo que gera um campo eletromagnético que alimenta o *Target* com a energia necessária para este poder estabelecer comunicação. É possível a comunicação entre dois dispositivos alimentados e com capacidade de gerar o campo eletromagnético, sendo que estes alternam os papeis de *Initiator* e *Target* repetidamente, de modo a garantir a comunicação nos dois sentidos.

Um dispositivo *Initiator* é geralmente um *smartphone*, PC ou qualquer outro dispositivo que não possua restrições em termos de energia e memória, sendo que um *Target* é geralmente um elemento passivo sem alimentação e com pouca memória (tipicamente menos que 4 MBytes), como é caso de uma *tag*, cartão ou autocolante. Este tipo de dispositivo apresenta também um baixo custo e é geralmente visto como

descartável ou facilmente substituível.

A tecnologia NFC é geralmente utilizada nos seguintes contextos:

- Pagamento com cartões de crédito eletromagnéticos, de modo a poderem ser utilizados sem necessidade de contacto físico;
- Pagamento por via de sistemas de *e-Wallet*, que permitem o acesso a informação bancária a partir de comunicação NFC;
- Sistemas de transporte público;
- Emparelhamento de dispositivos Bluetooth, sem a necessidade de descoberta de dispositivos e introdução de PIN;
- Controlo de acesso a viaturas por via informação armazenada numa *tag* NFC;
- Recolha de dados médicos de pequenos dispositivos.

Tal como já foi explicado, existem alguns tipos de codificação e modulação utilizados nas tecnologias NFC [4], de modo a permitir diferentes níveis de desempenho:

- NFC-A
  - Este tipo de comunicação permite taxas de 106 kbps, utilizando um esquema de codificação baseado no esquema de codificação de Miller, tal como visto na figura 2.6.

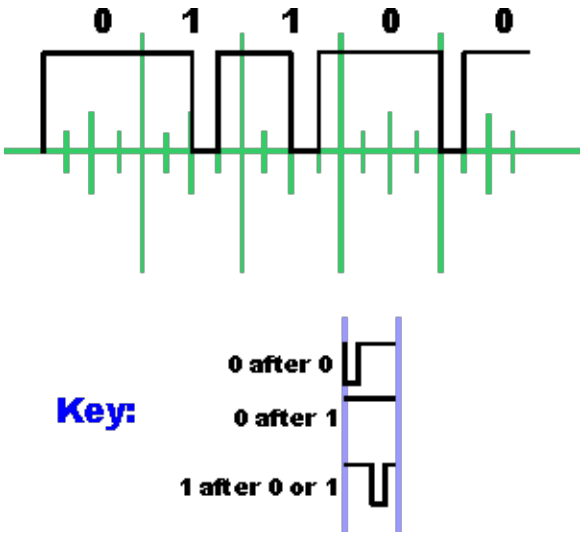


Figura 2.6: Codificação de Miller utilizada no NFC (imagem retirada de [4]).

- NFC-B
  - Este tipo de comunicação permite taxas de 212 kbps, utilizando o esquema de codificação de Manchester.

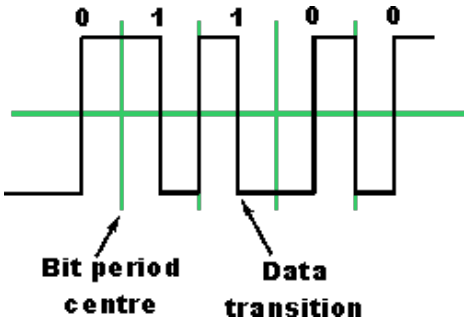


Figura 2.7: Codificação de Manchester utilizada no NFC (imagem retirada de [4]).

- NFC-F
  - Este tipo de comunicação permite taxas de 424 kbps, utilizando também o esquema de codificação de Manchester. Foi criado pela Sony e é geralmente denominado como FeliCa.

Também em termos de *tags* NFC, existem 4 tipos diferentes, que podem ser utilizados consoante as necessidades da aplicação em termos de memória ou taxa de transmissão dos dados:

- Tipo 1
  - Tipo de *tag* que utiliza NFC-A, permitindo a escrita e leitura de dados a uma taxa de 106 kbps. Em termos de memória, apresentam no mínimo 96 bytes, podendo chegar até aos 2 KB.
- Tipo 2
  - Tipo de *tag* que utiliza NFC-A, permitindo a escrita e leitura de dados a uma taxa de 106 kbps. Em termos de memória, apresentam no mínimo 48 bytes, podendo chegar até aos 2 KB.
- Tipo 3
  - Tipo de *tag* que utiliza NFC-F, permitindo a escrita e leitura de dados a uma taxa de 212 kbps. Em termos de memória, apresentam no mínimo 2 KB, podendo chegar até ao 1 MB.
- Tipo 4
  - Tipo de *tag* que utiliza NFC-A ou NFC-B, permitindo a escrita e leitura de dados a uma taxa de 106, 212 ou 424 kbps. Em termos de memória, podem chegar até aos 32 KB.

Para terminar, é apenas importante referir que está definido um modelo de dados utilizado para escrita de dados nas *tags* NFC, denominado NDEF (NFC Data Exhange Format), de modo a garantir a leitura uniforme dos dados armazenados numa *tag* NFC. Este formato é apresentado em detalhe na secção 5.1.3.1 do capítulo de implementação.

2.2 Sistemas de suporte ao AAL

É sempre interessante ter uma noção de sistemas semelhantes existentes no mercado. Nesta secção, é feita uma breve introdução a três sistemas de Ambient Assisted Living, contextualizando-os e comparando-os com o *CareStore*, de modo a que se perceba as diferenças, vantagens e desvantagens dos mesmos em relação ao *CareStore*.

Como será visível, a arquitetura da maioria dos sistemas deste género pouco difere.

2.2.1 SigHealth Platform

Esta plataforma, desenvolvida pela Signove[1], é talvez a mais parecida com a plataforma do projeto *CareStore*, como poderemos ver no capítulo seguinte, onde é apresentada a arquitetura do *CareStore*.

A SigHealth Platform apresenta três elementos principais na sua arquitetura: o Utilizador, o Monitor e o Médico. Um Utilizador recolhe os seus dados médicos recorrendo a uma série de dispositivos médicos com capacidades Bluetooth, sendo esses dados enviados automaticamente para a Internet através de uma equipamento compatível com o sistema (pode ser um *tablet* ou *smartphone*). O Monitor analisa as informações enviadas por um Utilizador e serve como intermediário entre o Médico e o Utilizador, comunicando com ambos de modo a garantir a eventual marcação de consultas presenciais. Relativamente ao Médico, este também tem acesso às informações enviadas pelo Utilizador e disponibiliza um diagnóstico mais preciso durante uma eventual consulta com o Utilizador.

Em termos de arquitetura, a figura 2.8 permite observar a arquitetura da SigHealth Platform.

[1] `http://www.signove.com/`

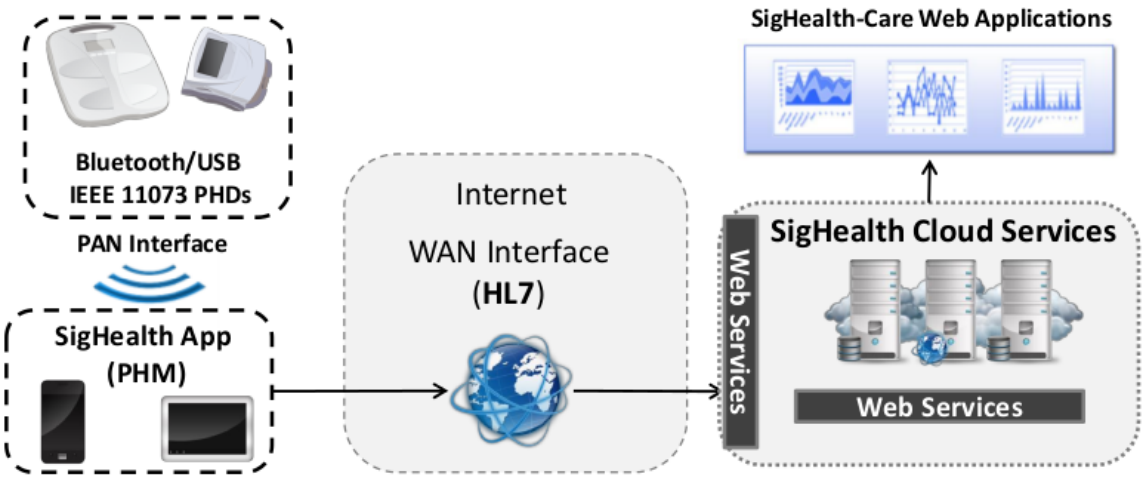


Figura 2.8: Arquitetura da SigHealth Platform (imagem retirada de [5]).

As diferenças mais assinaláveis desta plataforma com a plataforma do projeto *CareStore* tem a ver com a plataforma que serve de interface para a Internet. Nesta plataforma, é utilizada uma aplicação, acessível através do mercado de aplicações da Google[2], que executa o *software* necessário para recolher os dados dos dispositivos médicos, tratando também da identificação do utilizador que está a usar o sistema, para que os agentes de saúde possam saber a quem se referem os dados. No caso do *CareStore*, existe a possibilidade de ter várias aplicações que são descarregadas através de um *marketplace* criado especialmente para o *CareStore*. Relativamente ao resto da arquitetura, como será possível analisar, as semelhanças são bastante evidentes.

Em termos de dispositivos médicos a utilizar, a SigHealth Platform, tal como o *CareStore*, está preparada para dispositivos certificados pela Continua Alliance, garantindo assim a interoperabilidade entre dispositivos de diversos fabricantes.

2.2.2 Bosch Health Buddy System

Este sistema, desenvolvido pela secção Healthcare da Bosch[3], é um pouco diferente do *CareStore*, apesar de também consistir numa plataforma de AAL. No entanto, esta plataforma é constituída por *software* e *hardware* proprietário, o que torna todo o

[2] `https://play.google.com`
[3] `http://www.bosch-healthcare.com/`

sistema mais inflexível. A figura 2.9 apresenta a arquitetura da plataforma Health Buddy.

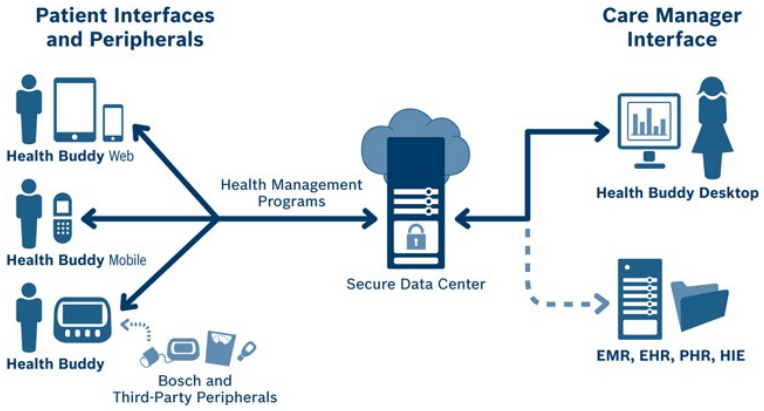


Figura 2.9: Arquitetura da plataforma Health Buddy System da Bosch (imagem retirada de [6]).

Como é possível ver, os utilizadores tem três tipos de interfaces diferentes para interagir com o sistema: Web, Mobile e um dispositivo chamado Health Buddy, que se trata de uma solução proprietária da Bosch.

A interface Web é composta por aplicações para *smartphone* e *tablet*, servindo para consultar ou registar dados de saúde no sistema online da Bosch. Também é recorrendo a esta interface que os utilizadores se registam e identificam com o sistema.

A interface Mobile inclui um telemóvel de uso fácil que permite a recolha de informação dos pacientes por via de uma interface de utilizador simples, para além de facilitar a comunicação com as autoridades de saúde.

O Health Buddy é um dispositivo instalado na casa do utilizador, permitindo a recolha de dados a partir de dispositivos de recolha de dados médicos, desenvolvidos pela Bosch ou por alguns fabricantes escolhidos pela Bosch. Os dispositivos desenvolvidos pela empresa possuem comunicação Bluetooth e são certificados pela Continua Alliance, de modo a garantir a interoperabilidade com dispositivos de outros fabricantes. Para além disso, a Bosch garante o suporte a diversos dispositivos de fabricantes diferentes, recolhendo os seus dados via USB. Para além disto, o Health Buddy implementa um sistema de interação com o utilizador baseado em perguntas, que permite a recolha de dados médicos dos mesmos e a partilha de informação e

bons hábitos de saúde.

Mencionados os dispositivos que fazem a interface com o utilizador, os dados recolhidos por estas interfaces são depois armazenados num servidor seguro, ficando acessíveis aos profissionais de saúde a partir de um *website*, de modo a que estes possam interpretar os dados recolhidos e agir de acordo com essa interpretação. Para além disto, os dados podem também ser acedidos por entidades de saúde autorizadas, através de *web services* desenvolvidos pela Bosch.

Resumindo, este sistema tem como desvantagem a obrigação de utilizar *hardware* proprietário, o que restringe bastante o utilizador em termos do equipamento que pode comprar. O mesmo acontece com o *software*, visto que não existe maneira de implementar aplicações personalizadas para o Health Buddy System. Para além disso, o Health Buddy não possui uma interface muito acessível, na medida em que obriga à interação com o utilizador de modo a recolher alguns dos dados. A demonstração do funcionamento do Health Buddy disponibilizada pela Bosch deixa isto bem claro[4].

No entanto, trata-se de uma solução madura de suporte aos objetivos do AAL.

2.2.3 Google Fit

O Google Fit[5], desenvolvido pela Google, trata-se de um sistema de monitorização de parâmetros que indiquem a condição física do utilizador. Apesar deste sistema não ser desenvolvido com o objetivo de controlar os dados de saúde dos pacientes, pode ser visto como uma plataforma que cumpre alguns dos objetivos do AAL.

Os objetivos do Google Fit são:

- Permitir aos utilizadores armazenar e aceder a informação relativa à sua condição física;
- Permitir a criação de aplicações que utilizem a informação armazenada nos servidores do Google Fit;

[4] `http://www.bosch-telehealth.com/en/us/products/health_buddy/health_buddy_demo.html`

[5] `https://fit.google.com/`

- Garantir a integração de qualquer tipo de dispositivo sensor.

Ou seja, apesar do Google Fit ser mais utilizado com o objetivo de monitorizar a condição física dos seus utilizadores, a Google permite o acesso aos dados recolhidos, por parte através de API's criadas para esse efeito. Isto permite a criação de aplicações de saúde, visto que também é um dos objetivos da Google a integração de qualquer aparelho sensor. Resumindo, são dadas todas as ferramentas necessárias ao programador para aumentar o alcance da plataforma, permitindo-a ser utilizada com finalidades que a Google neste momento não cobre.

Em termos de arquitetura da plataforma, a figura 2.10 ilustra os componentes da plataforma Google Fit.

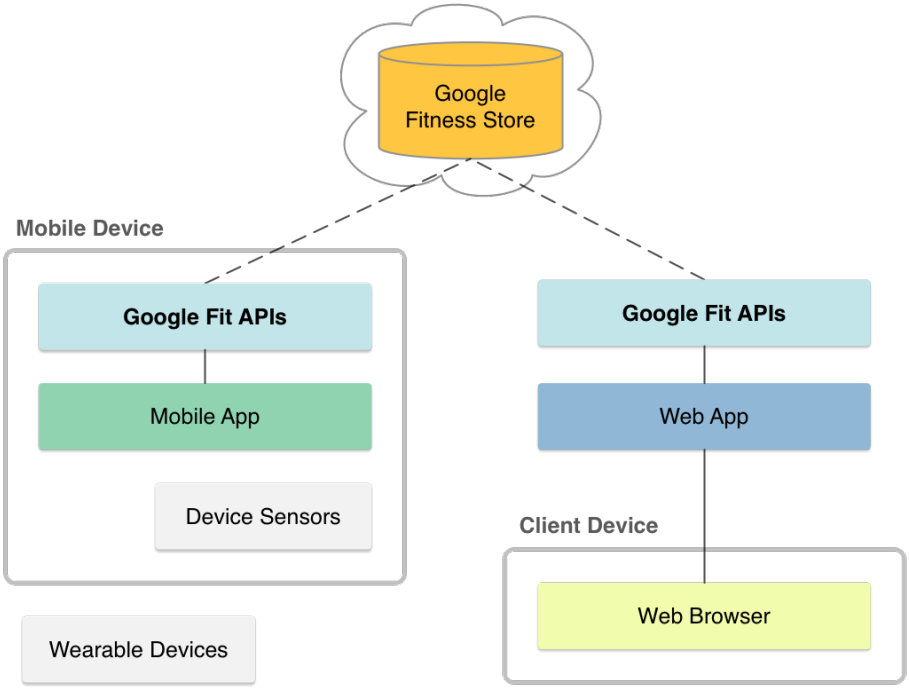


Figura 2.10: Arquitetura da plataforma Google Fit (imagem retirada de [7]).

Basicamente, existem dois métodos principais de interagir com a plataforma. Através de uma aplicação para *smartphones* com o sistema operativo Android, desenvolvida também pela Google. Esta aplicação permite a utilização dos sensores do *smartphone* em questão e a comunicação com dispositivos sensores externos, para

recolha de dados. Também é possível a utilização de um *website*, que será um pouco mais limitado, na medida em que não permite a comunicação com *wearables* ou dispositivos sensores.

Os dados recolhidos através de qualquer uma destas interfaces são armazenados na Google Fitness Store, passando a estar acessíveis, mediante as autorizações definidas pelo utilizador responsável por esses dados.

De modo a permitir o desenvolvimento de aplicações que utilizem os dados armazenados na Google Fitness Store, foram desenvolvidas duas API's:

- Android Fit API
  - Esta API permite o desenvolvimento de aplicações para a plataforma Android, facilitando a recolha e interpretação de dados de dispositivos sensores. Para além disso, pode ser utilizada em conjunto com as restantes API do sistema operativo Android, para tirar o máximo partido da plataforma. Por exemplo, pode ser utilizada a API Bluetooth *Low Energy* para ler dados de sensores que utilizem este tipo de comunicação.
- REST API
  - Esta API é a que dá flexibilidade ao sistema. Com o uso da API REST, passa a ser possível a criação de aplicações para outras plataformas que não o Android, tornando o Google Fit apelativo para um maior número de utilizadores.

Portanto, o Google Fit, apesar de não ser criado com o objetivo da recolha de dados médicos, pode ser utilizado para esse fim, na medida em que apresenta uma plataforma bastante flexível e disponibiliza API's pensadas para esse efeito. Cabe aos programadores decidirem a finalidade com que pretendem utilizar a plataforma.

Capítulo 3

Especificação do Sistema

Neste capítulo será feita a especificação do sistema desenvolvido no projeto *CareStore*, de modo a dar a conhecer a arquitetura do sistema, os seus elementos e as suas características. Após a leitura deste capitulo, deverá ser possível ter uma visão bastante geral das componentes do sistema e entender qual o papel de cada um desses componentes.

3.1 Arquitetura do Sistema

O projeto *CareStore* surge após algumas tentativas passadas de implementar plataformas de Ambient Assisted Living. Algumas dessas plataformas ou ferramentas como o OpenCare [24], OpenAAL [25] ou UniversAAL [26], acabaram por nunca passar de tentativas que não contaram com adesão significativa por parte das entidades de saúde europeias. O *CareStore* tentou aprender com os erros ou vulnerabilidades destas plataformas e aproveitou alguns dos pontos fortes destas plataformas, de modo a criar uma plataforma mais atraente.

Tendo isto, a figura 3.1 apresenta uma visão geral da arquitetura do *CareStore* e quais os módulos que é possível que compõem a mesma.

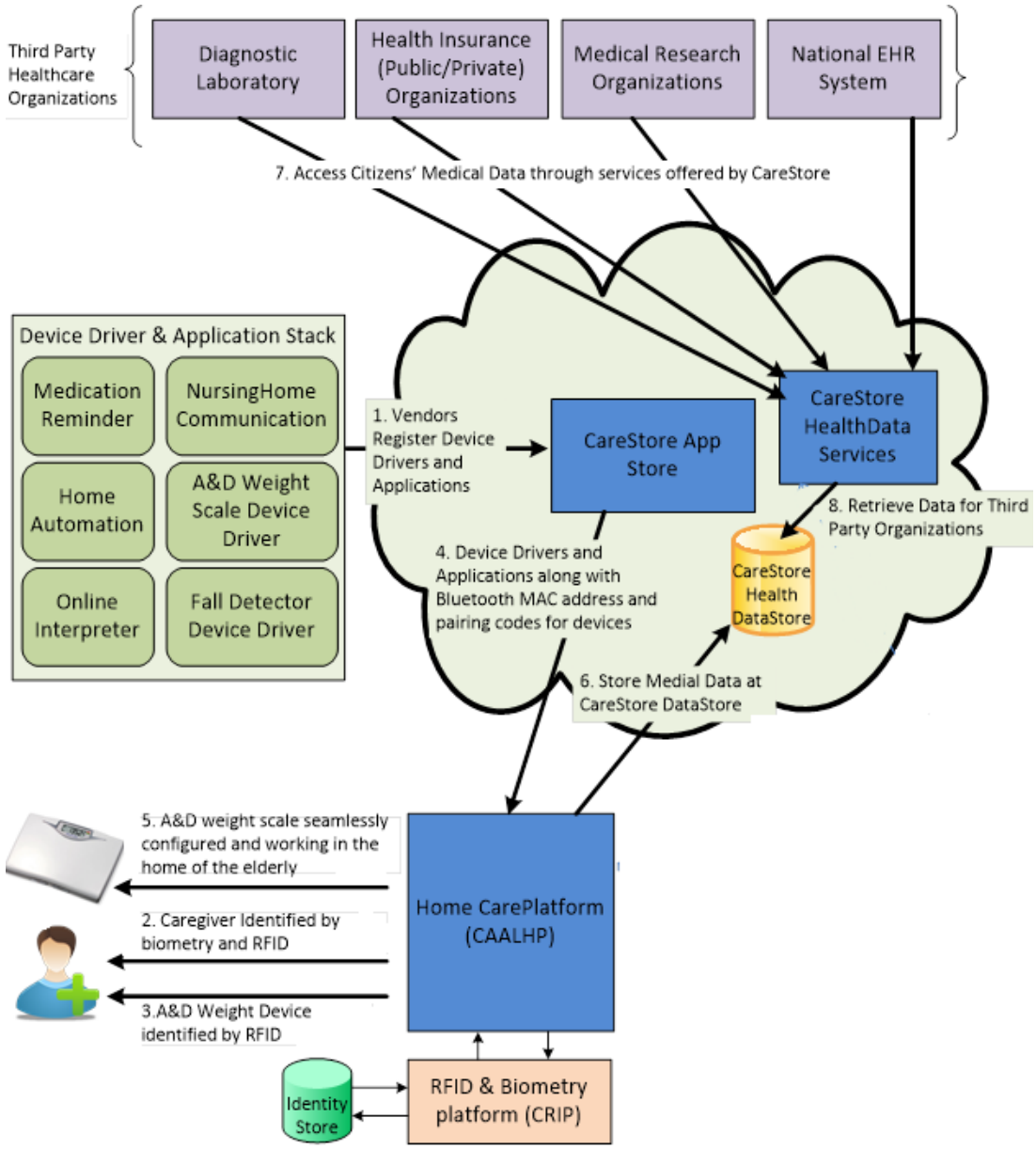


Figura 3.1: Arquitetura do *CareStore* (imagem retirada de [8]).

É um pouco difícil entender a figura 3.1 sem que esta seja explicada minimamente.

Basicamente a arquitetura do *CareStore* é dividida em três grandes entidades. Começando de baixo para cima, temos a Common Ambient Assisted Living HomePlatform (CAALHP) e a Common Recognition and Identification Platform (CRIP) que compõem o equipamento que é efetivamente instalado no local onde a plataforma

CareStore é instalada. É através destas plataformas que os assistentes de saúde e utilizadores críticos (idosos, pessoas com deficiência, etc) poderão interagir com o sistema *CareStore*, usufruindo das suas funcionalidades a nível de recolha de dados médicos. Do ponto de vista do utilizador, a interação com o sistema é feita com CAALHP, visto que este é o elemento que implementa a interface com os utilizadores. Para além disto, esta plataforma é que faz a ponte entre o sistema instalado no local e os serviços instalados na *cloud* da plataforma *CareStore*. Relativamente ao CRIP, este trata de aspetos puramente funcionais do sistema instalado no local, como será explicado. Para além disto, são também utilizados dispositivos médicos para interagir com a plataforma instalada na habitação dos utilizadores. Estes dispositivos não estão restringidos permanentemente a uma localidade, podendo ser transportados e utilizados em diferentes locais, desde que a plataforma *CareStore* esteja instalada nesse mesmo local e os dispositivos sejam identificados e validados pelo CRIP instalado no local. Na figura, isto é exemplificado com uma balança fabricada pela A&D Medical[1].

De modo a garantir o fácil acesso à informação médica dos utilizadores da plataforma *CareStore*, esta implementa algumas funcionalidades e serviços armazenados na *cloud*. Talvez a componente mais importante seja a *CareStore App Store* de onde é possível descarregar aplicações de saúde e *drivers* de dispositivos médicos, consoante as necessidades de cada local onde é instalado o sistema descrito anteriormente, composto por CAALHP e CRIP. Para além disto, na *cloud* são disponibilizados serviços que permitem o acesso aos dados médicos recolhidos por parte de entidades externas, desde que estas sejam autorizadas.

Tal como já foi dito, na habitação dos utilizadores, são instalados os subsistemas CAALHP e CRIP. Estes sistemas tem funcionalidades bastante diferentes, no entanto são ambos essenciais para que a experiência *CareStore* do utilizador seja completa. É importante identificar as funcionalidades destas entidades separadamente.

- CAALHP
  - A Common Ambient Assisted Living Home-Platform é um sistema *open source*, desenvolvido pela Universidade de Aarhus e que garante a interface com os utilizadores, para além de permitir a instalação de aplicações de

[1] `http://www.andonline.com/`

saúde, a partir do *marketplace* de aplicações instalado nos serviços de *cloud* do *CareStore*. Este subsistema comunica com o CRIP de modo a implementar todas as funcionalidades que compõem o sistema instalado na habitação dos utilizadores. Também é apenas através deste subsistema que é feito o acesso aos serviços *cloud* do *CareStore*;

- CRIP
  - A Common Recognition and Identification Platform é também um sistema *open source*, desenvolvido pela Universidade do Minho, possibilitando a identificação de utilizadores, funcionários de saúde e dispositivos médicos através de tecnologias Near Field Communication (NFC) e reconhecimento biométrico. Para além disto, este sistema implementa a comunicação com os dispositivos médicos de modo a garantir a recolha automática de dados médicos dos utilizadores do sistema *CareStore*. O CRIP não estabelece comunicação direta com os serviços *cloud* do *CareStore*, comunicando diretamente com o CAALHP.

Para além disto, existe também o *marketplace* de aplicações do *CareStore*, uma plataforma *online* que irá permitir que os utilizadores descarreguem facilmente, a partir do CAALHP, as aplicações mais adequadas ao local de instalação do sistema *CareStore*. O tipo de aplicações mais interessantes num sistema deste género são naturalmente as de saúde, mas nada impede a criação e disponibilização de outro tipo de aplicações no *marketplace*. Esta entidade é provavelmente o ponto que distingue o *CareStore* da maioria dos outros projetos relacionados com AAL. A possibilidade de criar aplicações especializadas para uma certa necessidade garante uma flexibilidade tremenda ao sistema, não forçando o utilizador a apenas um tipo específico de *hardware* e permitindo personalização, no sentido em que o utilizador é que escolhe as aplicações que pretende utilizar.

Outro pormenor extremamente importante do sistema *CareStore*, tal como visível pela imagem 3.1, é a possibilidade de organizações de saúde externas poderem aceder aos serviços oferecidos pela infraestrutura *cloud* do *CareStore*. Isto permite o rápido acesso aos dados médicos dos utilizadores do *CareStore*, por parte de autoridades de saúde autorizadas. Naturalmente, devido à sensibilidade da informação que é transmitida no sistema *CareStore*, os processos de comunicação entre as entidades

do sistema implementam mecanismos de segurança, de modo a proteger o acesso ao sistema e aos dados. No entanto, isso está fora do âmbito deste documento.

Visto este documento assentar sobre o trabalho desenvolvido pela Universidade do Minho no CRIP, é importante explicar melhor este subsistema.

3.1.1 CRIP

Tal como já foi mencionado, os objetivos do CRIP são os seguintes:

- Implementação de um mecanismo de identificação de utilizadores e dispositivos médicos, com recurso a tecnologias NFC e reconhecimento biométrico;
- Recolha de dados dos dispositivos médicos, via Bluetooth;
- Garantir ao CAALHP a integração de dispositivos médicos de modo transparente;
- Garantir a segurança dos processos de comunicação com o CAALHP e dispositivos médicos.

Apresentados os objetivos da plataforma CRIP, esta interage com as entidades identificadas na imagem 3.2. Ou seja, é responsabilidade do CRIP a identificação de utilizadores do sistema (quer profissionais de saúde, quer beneficiários do sistema) e a identificação e comunicação com os dispositivos médicos. Para além disto, esta plataforma não tem acesso direto aos serviços *cloud* do *CareStore*, tendo toda a informação de ser enviada ao CAALHP antes de sair do local onde o sistema se encontra instalado.

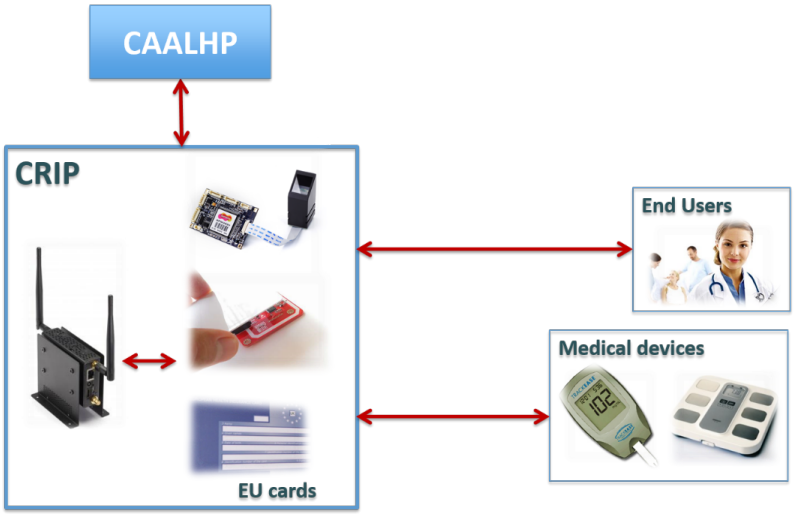


Figura 3.2: Conceito do subsistema CRIP.

De modo a atingir os objetivos propostos para este subsistema, ele apresenta as seguintes características em termos de *hardware* e *software*.

- Sistema operativo baseado em Linux;
- Suporte de *hardware* e *software* para Bluetooth 4.0;
- Suporte de *hardware* e *software* à tecnologia NFC;
- Suporte de *hardware* e *software* a dispositivos biométricos;
- Conexão e configuração de ligação Ethernet ao CAALHP;
- Disponibilização de uma API de acesso remoto, através de pedidos HTTP.

Em termos de arquitetura de *software*, o CRIP está organizado tal como explicado na imagem 3.3.

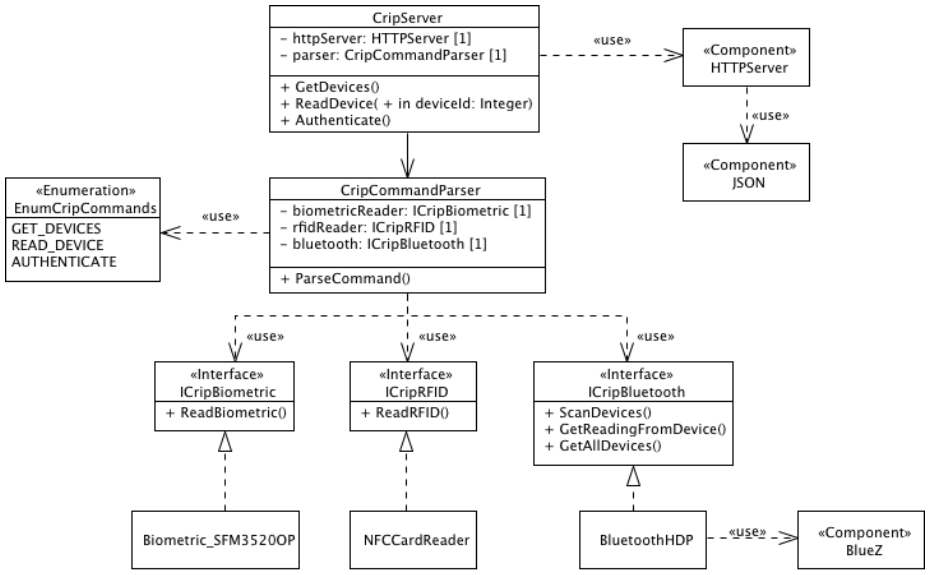


Figura 3.3: Arquitetura de *software* do CRIP.

A figura 3.3 expõe, de forma bastante resumida, a arquitetura de *software* do CRIP. Resumidamente, existem quatro grandes componentes. Primariamente, é implementado um servidor HTTP e de *parsers* que permitem a receção e interpretação de ordens de acesso às funcionalidades do CRIP, garantindo também a comunicação com o CAALHP. As funcionalidades do CRIP são implementadas com recurso aos outros três módulos: módulo Biométrico, módulo NFC e módulo Bluetooth. O módulo Biométrico implementa a identificação e registo de utilizadores (idosos e funcionários de saúde). O módulo NFC implementa a identificação de utilizadores e dispositivos médicos. Como será possível ver no capítulo 5, este módulo também é utilizado no processo de emparelhamento de dispositivos Bluetooth implementado no *CareStore*. O módulo Bluetooth implementa a descoberta e emparelhamento de dispositivos Bluetooth e a comunicação com dispositivos médicos.

Tal como já foi mencionado, a Universidade do Minho apenas está responsável pelo desenvolvimento do CRIP e pela integração deste com o CAALHP. Relativamente a este documento, este não cobre toda a implementação do CRIP, referindo-se apenas aos passos relacionados com o desenvolvimento de funcionalidades Bluetooth deste subsistema, que representam uma componente bastante importante do projeto

CareStore.

3.2 Tecnologias e ferramentas utilizadas

Para ir de encontro aos objetivos definidos na secção anterior, foi necessário o uso de ferramentas de *hardware* e *software* que ajudassem ao cumprimento dos mesmos. Nesta secção será feita uma apresentação das ferramentas utilizadas no CRIP de modo a implementar as funcionalidades presentes neste subsistema. Em termos de *hardware*, são apresentados os módulos que compõem o subsistema CRIP, enquanto que, relativamente ao *software*, apenas são apresentadas as ferramentas utilizadas no desenvolvimento de funcionalidades Bluetooth.

Segue-se a apresentação das ferramentas utilizadas.

3.2.1 Hardware

O CRIP é composto por vários módulos de *hardware*. Nesta secção é feita uma breve introdução aos mesmos, de modo a que se perceba o que é possível encontrar, a nível físico, nesta plataforma.

Raspberry Pi Model B+

Figura 3.4: Raspberry Pi Model B+.

A Raspberry Pi[2] trata-se de um computador de baixo custo e de pequeno tamanho. O modelo B+, utilizado no CRIP, possui um processador ARM de 700MHz e 512MB de memória RAM. Está equipado com uma porta RJ45, que permite conexão Ethernet. Para além disto, possui 4 entradas USB 2.0. No projeto *CareStore*, esta é a peça central do *hardware*.

Leitor ACR122 USB NFC

Figura 3.5: ACR122 USB NFC Reader.

Este leitor NFC da Advanced Card Systems Ltd[3] permite a leitura e escrita de dispositivos NFC ISO 14443 A e B, Mifare, FeliCa e os 4 tipos de *tag* NFC compatíveis com a norma ISO/IEC18092. A interface com o dispositivo que irá receber os dados é feita por via de um USB integrado. É este dispositivo que garante o suporte às funcionalidades NFC do CRIP.

[2] `http://www.raspberrypi.org`
[3] `http://www.acs.com.hk/`

Módulo Bluetooth Smart Ready BT111

Figura 3.6: BT111, módulo Bluetooth Smart Ready.

Este módulo Bluetooth *Smart Ready* da Bluegiga[4] garante o suporte de *hardware*das funcionalidades Bluetooth do CRIP. O BT111 implementa um mecanismo *dual radio* que permite a comunicação através de Bluetooth *Classic* e Bluetooth *Low Energy* até 100 metros. Para além disto, possui uma interface HCI de modo a ser compatível com várias *stacks* Bluetooth existentes no mercado. É fornecido numa placa de desenvolvimento com uma interface USB, de modo a ser facilmente integrado com qualquer sistema.

Módulo Biométrico SFM3520-OP

Figura 3.7: Módulo Biométrico Suprema SFM3520-OP.

Este módulo biométrico desenvolvido pela Suprema[5], permite a leitura de impressões digitais através de um sensor ótico de alta qualidade. Está equipado com um algoritmo que garante a correspondência rápida de impressões digitais (até 970

[4] `https://www.bluegiga.com`
[5] `http://www.supremainc.com/`

milissegundos), e permite o armazenamento de, no máximo, 9000 impressões digitais.

Tags NFC NTAG203

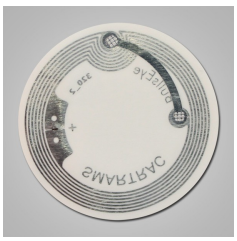


Figura 3.8: NTAG203 NFC Tags

Desenvolvidas pela NXP Semiconductors[6], foram as *tags* NFC utilizadas no desenvolvimento do projeto. Sendo *tags* do tipo 2, podem armazenar 144 bytes. Operam na frequência de 13,56 MHz e permitem velocidades de transferência de 106kbps. No projeto *CareStore* estas *tags* são usadas nos dispositivos médicos, contendo a informação necessária para os emparelhar via Bluetooth com o CRIP.

Dispositivos médicos

(a) UA-767PBT-C - Blood Pressure Monitor

(b) UC-321PBT-C - Weighing Scale

Figura 3.9: Dispositivos médicos utilizados no desenvolvimento do *CareStore*.

Os dispositivos médicos, apresentados na figura 3.9, fabricados pela A&D Medical[7], representam os dispositivos utilizados ao longo do desenvolvimento do *CareStore*. Ambos são certificados pela Continua Alliance, e possuem capacidade de exportar as

[6] `http://www.nxp.com/`
[7] `http://www.andonline.com/`

amostras recolhidas via Bluetooth *Classic*.

O `UA-767PBT-C` trata-se de um medidor de pressão sanguínea e permite o armazenamento de 40 medições que podem ser transferidos via Bluetooth assim que isso for possível.

O `UC-321PBT-C` é uma balança que suporta até 200kg com uma resolução de 100g. Permite o armazenamento de 40 amostras que podem ser transferidas via Bluetooth assim que possível.

3.2.2 Software

De modo a aproveitar as capacidades do *hardware* apresentado na secção anterior, foi necessário utilizar várias ferramentas de desenvolvimento de *software*. Nesta secção, apenas serão apresentadas as ferramentas de *software* relacionadas com as funcionalidades Bluetooth do sistema, relevantes para o resto do documento.

BlueZ

A BlueZ [27] é uma *stack* Bluetooth utilizada em sistemas operativos Linux. O seu objetivo é a implementação da especificação Bluetooth. Apresenta como características e/ou vantagens:

- *Open source*;
- Implementação modular;
- Abstração relativamente ao *hardware*, dando suporte a bastantes dispositivos;
- Dá suporte ao Bluetooth *Classic* e *Low Energy*;
- Implementação da *Host Controller Interface* (HCI), de modo a poder estabelecer comunicação entre Host (BlueZ) e Controller (BT111) Bluetooth, tal como explicado na secção 4.1;
- Implementação de bastantes perfis Bluetooth, nomeadamente o HDP, que é essencial para a recolha de dados médicos de dispositivos.

Esta *stack* interage com o módulo BT111, recorrendo a uma interface HCI, tal como será explicado neste documento. A versão utilizada no desenvolvimento do *CareStore* foi a versão `4.101`.

DBUs

O DBus[8] trata-se de um mecanismo de comunicação interprocessos implementa um barramento de mensagens que é utilizado como uma maneira simples de permitir que aplicações comuniquem entre si. Este mecanismo é utilizado pela BlueZ de modo a permitir a invocação de operações sobre a *stack* Bluetooth, como por exemplo a descoberta e o emparelhamento de dispositivos. No projeto *CareStore* foi utilizada a versão `1.6.12`.

Antidote

O Antidote[9] é uma implementação dos *standards* IEEE 11073, que permite a comunicação com dispositivos médicos. Estes *standards* implementam os modelos de dados dos dispositivos médicos, o que permite a interoperabilidade entre dispositivos de diferentes fabricantes. A versão utilizada no projeto *CareStore* foi a versão `2.0`.

libNFC

A libNFC [10] é uma *stack* NFC *open source*, disponível para os principais sistemas operativos. No caso do *CareStore*é utilizada em conjunto com o leitor NFC ACR122, de modo a garantir a possibilidade de comunicação com dispositivos NFC. A versão utilizada no projeto *CareStore* é a `1.7.1`.

[8] `http://www.freedesktop.org/wiki/Software/dbus/`
[9] `http://oss.signove.com/index.php/Antidote:_IEEE_11073-20601_stack`
[10] `http://nfc-tools.org/index.php?title=Libnfc`

Capítulo 4

Introdução à Programação com BlueZ

No caso do projeto *CareStore*, como o uso de *software open source* se trata de uma imposição, a pilha de *software* utilizada, tal como já foi mencionado, é a BlueZ, que se trata da pilha Bluetooth adotada oficialmente por sistemas Linux. Esta secção deverá ser vista como um guia introdutório à programação usando a extensa API fornecida pela pilha Bluetooth BlueZ. A documentação disponível relativamente a esta matéria é escassa e pouco completa. Algo que também é bastante importante de referir é que a versão da BlueZ usada no projeto *CareStore* é a versão 4 e que há diferenças significativas entre esta versão e a mais recente, a 5 [28], que deverão ser tidas em conta no caso de se querer adotar esta versão.

Antes de se falar da implementação de funcionalidades Bluetooth, é importante salientar o sistema que podemos encontrar no *CareStore*. Como já foi mencionado, o *hardware* escolhido para o projeto foi o módulo BT111 da Bluegiga [29]. Este módulo apenas implementa as camadas mais baixas da pilha Bluetooth, comunicando com a pilha de *software* BlueZ, através de uma camada HCI via USB. A imagem 4.1 demonstra melhor aquilo que foi dito.

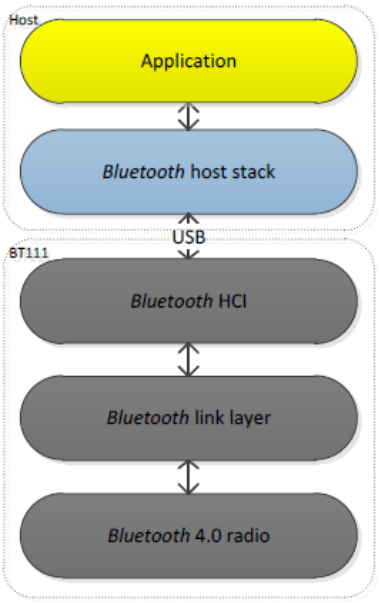


Figura 4.1: Arquitetura Host/Controller do *CareStore* (imagem retirada de [9]).

De modo a que se perceba melhor os conceitos de *Host* e *Controller*, será importante a leitura da secção 4.1.

O HCI permite a comunicação entre *Host* e *Controller* Bluetooth, de modo a permitir ao *Host* ter acesso às funcionalidades fornecidas pelo *Controller*. Transpondo isto para o caso do projeto *CareStore*, permitiria à BlueZ ter acesso às funcionalidades do BT111. Naturalmente, a BlueZ disponibiliza uma implementação quase completa deste protocolo via USB, o que permite ao programador estabelecer comunicação com qualquer tipo de *hardware* Bluetooth que implemente o módulo HCI.

Para além desta API, a BlueZ disponibiliza funcionalidades através de um mecanismo denominado DBus. Esta API garante uma maior abstração relativamente ao HCI e torna o código desenvolvido mais compreensível e compacto.

4.1 HCI

Antes de avançar, é importante fazer uma pequena introdução ao HCI.

Esta interface é necessária devido à separação da pilha Bluetooth em duas partes, o que acontece de modo a permitir a abstração das camadas mais elevadas da pilha, relativamente ao *hardware* Bluetooth utilizado. Ou seja, uma aplicação desenvolvida

utilizando a API da BlueZ deverá funcionar independentemente do *hardware* utilizado, desde que este implemente a camada HCI. Esta interface trata de interligar dois módulos diferentes, o *Host* e o *Controller*.

O *Host* implementa as camadas mais elevadas da pilha Bluetooth, não se preocupando com o aspeto físico da especificação Bluetooth, tal como visível na figura 4.1. No projeto *CareStore*, o papel de *Host* é garantido pela pilha BlueZ. Tal como já foi referido, a BlueZ implementa uma camada HCI via USB.

O *Controller*, tal como visível na figura 4.1, é responsável por implementar as camadas mais baixas da pilha Bluetooth, mais ligadas aos aspetos físicos da comunicação Bluetooth. No projeto *CareStore*, o módulo BT111 da Bluegiga representa o *Controller* e também implementa uma camada HCI via USB.

O HCI é composto por comandos que são transportados através daquilo que se denomina Host Controller Transport Layers. É importante não confundir as duas entidades. O módulo HCI define os comandos que são trocados entre *Controller* e *Host*, enquanto que a camada de transporte define as regras relativamente à tecnologia de transporte que é utilizada para trocar esses comandos. Por exemplo, o módulo BT111 implementa HCI sobre USB. No entanto, existem módulos disponíveis no mercado que o fazem via UART. Ou seja, a implementação de uma camada de transporte garante ao sistema transparência entre o protocolo HCI propriamente dito e a tecnologia de transporte usada para garantir a troca de comandos HCI entre *Controller* e *Host* Bluetooth.

Relativamente aos comandos propriamente ditos, estes estão separados em grupos lógicos [10], consoante a sua função:

Comandos e Eventos HCI	
Eventos Genéricos	Eventos que ocorrem devido a múltiplos comandos ou eventos que podem ser disparados a qualquer momento
Configuração de dispositivo	Comandos que servem para forçar um *Controller* a entrar num estado predefinido
Controlo de Fluxo do *Controller*	Comandos e Eventos que servem para fazer controlo de fluxo de dados do *Host* para o *Controller*
Informação do *Controller*	Comandos que permitem ao *Host* a descoberta de informação do *Controller*
Configuração do *Controller*	Comandos e Eventos que permitem a visualização e configuração de parâmetros do *Controller*
Descoberta de dispositivos	Comandos e Eventos que permitem a descoberta de dispositivos Bluetooth na área circundante
Estabelecimento de conexão	Comandos e Eventos que permitem o estabelecimento de uma conexão com outro dispositivo
Informação Remota	Comandos e Eventos que permitem a descoberta de informação de um dispositivo remoto
Conexões Síncronas	Comandos e Eventos que permitem a criação de conexões síncronas
Estado da conexão	Comandos e Eventos que permitem a configuração de uma ligação. Especialmente importante no modo de operação *Low Power*
Estrutura da *Piconet*	Comandos e Eventos que permitem a descoberta e configuração da *Piconet*
Qualidade de Serviço	Comandos e Eventos que permitem especificar parâmetros de qualidade de serviço
Ligações Físicas	Comandos e Eventos que permitem a configuração de uma ligação física
Controlo de Fluxo do *Host*	Comandos e Eventos que permitem o uso de controlo de fluxo de dados no sentido do *Host*
Informação da ligação	Comandos e Eventos que permitem o acesso a informação relativa a uma ligação
Autenticação e Cifragem	Comandos e Eventos que permitem a autenticação de um dispositivo remoto e cifragem dos dados numa dada ligação
Teste	Comandos e Eventos que permitem que o dispositivo entre em modo de teste

Tabela 4.1: Tipos de Comandos e Eventos HCI (adaptado de [10]).

Para uma consulta detalhada relativamente aos comandos e eventos disponíveis, será melhor a consulta direta da especificação do protocolo HCI, na medida em que

se torna impossível fazer essa análise nesta tese, devido à quantidade de informação existente.

Algo que convém ser explicado sobre tabela 4.1 é a diferença entre um Evento e um Comando. A diferença entre os dois baseia-se, basicamente, no sentido da comunicação e na entidade que gera a informação.

Um Comando é enviado do *Host* para o *Controller*, no sentido de fazer um pedido ou alguma configuração ao *hardware*.

Um Evento é enviado do *Controller* para o *Host* e em termos de informação pode conter os dados de uma notificação por parte do *Controller* ou representar uma resposta a um pedido do *Host*.

Para terminar esta pequena introdução, é interessante ter uma noção dos pacotes que são trocados usando o protocolo HCI. Temos quatro tipos de pacotes:

- HCI Command Packet;
- HCI ACL Data Packet;
- HCI Synchronous Data Packet;
- HCI Event Packet.

HCI Command Packet

Este tipo de pacote é usado no envio de comandos do *Host* para o *Controller*. Um *Controller* terá de estar preparado para receber pacotes deste tipo até 255 bytes, excluindo o *header*. Os diferentes tipos de comandos são identificados com recurso a um *opcode*.

A estrutura do pacote é apresentada na figura 4.2.

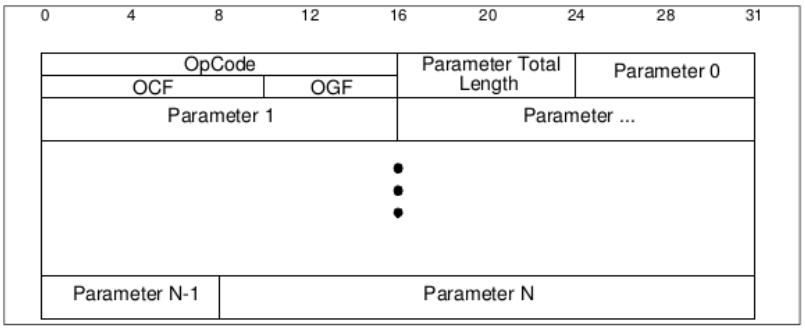


Figura 4.2: HCI Command Packet (imagem retirada de [10]).

Como é visível na imagem 4.2, o *opcode* é dividido em dois campos:

- **OGF** (OpCode Group Field) - Os primeiros 6 bits do *opcode* servem para identificar o grupo em que se insere o comando, de acordo com a sua funcionalidade. Para além disto, é possível a definição de comandos específicos de um dado fabricante;
- **OCF** (OpCode Command Field) - Os últimos 10 bits do *opcode* permitem identificar o comando propriamente dito.

Os outros campos são bastante simples de compreender. O campo *Parameter Total Length* ocupa 1 byte e indica o número de bytes ocupados pelos parâmetros. Relativamente aos parâmetros, estes apenas têm de obedecer à condição de ocupar um número inteiro de octetos. Não é necessária nenhuma separação entre parâmetros, pois os comandos são todos descriminados na especificação.

Para terminar, um exemplo de um comando HCI é apresentado na tabela 4.2. Nesta tabela, não se pretende apresentar a estrutura do pacote tal como visível na figura 4.2. O objetivo é apenas apresentar um exemplo do tipo de dados que é possível encontrar num HCI Command.

Comando	OGF	OCF	Parametros (n bytes)	Parametros Retorno
HCI_Inquiry	0x01	0x0001	LAP (3); Inquiry_Length (1); Num_Responses (1)	Não tem

Tabela 4.2: HCI Inquiry Command.

HCI ACL Data Packet

Estes pacotes servem para trocar dados entre *Host* e *Controller*. Existem dois subtipos de pacotes deste tipo: Automatically-Flushable e Non-Automatically-Flushable. Como a própria designação indica, a diferença entre os dois é a possibilidade de descartar os pacotes automaticamente ou não, após um certo período de tempo. O formato deste tipo de pacotes é apresentado na figura 4.3.

Figura 4.3: HCI ACL Data Packet (imagem retirada de [10]).

Como é possível observar, a estrutura é bastante simples. O campo *handle* permite identificar a conexão sobre a qual é feita a troca de dados, sendo que é possível ter algum controlo sobre como é realizado o processo de comunicação, modificando os dados dos campos de controlo que se seguem. Para além disto, apenas são enviados os dados propriamente ditos. Do ponto de vista do programador, este tipo de pacote não é muito importante, daí não ser necessário entrar em mais detalhe.

HCI Synchronous Data Packet

Este tipo de pacote serve para efetuar troca síncrona de informação entre *Host* e *Controller*. A estrutura e finalidade deste tipo de pacote é comparável à estrutura de HCI ACL Data Packets. O formato deste tipo de pacote é apresentado na figura 4.4.

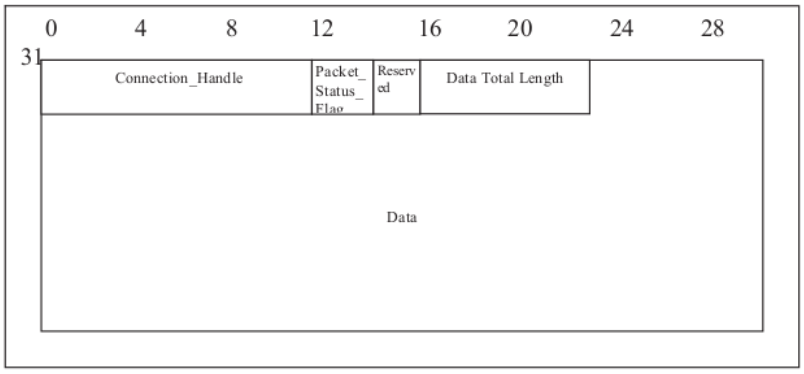


Figura 4.4: HCI Synchronous Data Packet (imagem retirada de [10]).

Este tipo de pacote é apresenta uma estrutura bastante simples. O campo *Connection_Handle* permite identificar a conexão sobre a qual é feita a troca de dados, sendo esta seguida de um conjunto de *flags*, utilizadas para controlo do processo de comunicação. Aparte disto, apenas é enviado o tamanho dos dados e os dados propriamente ditos.

HCI Event Packet

Este tipo de pacote é usado pelo *Controller* de modo a notificar o *Host* de eventos ocorridos. O tamanho máximo de dados que pode transportar é 255 bytes, sendo a sua estrutura bastante semelhante a pacotes HCI Command. A estrutura deste tipo de pacote é apresentada na figura 4.5.

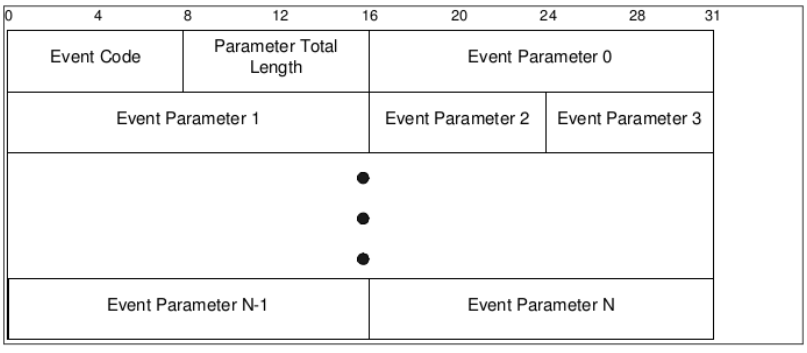


Figura 4.5: HCI Event Packet (imagem retirada de [10]).

Como é possível ver na imagem acima, a identificação do tipo de evento é ga-

rantida pelo campo Event Code, cujo tamanho é 1 byte. Para além disto, apenas temos um octeto, denominado Parameter Total Length, que nos indica o número de octetos ocupado pelos parâmetros. A estrutura dos parâmetros segue a mesma lógica apresentada relativamente aos pacotes HCI Command.

Para terminar, resta apenas apresentar um exemplo deste tipo de pacote. Neste caso, o exemplo apresentado na tabela 4.3 seria a resposta do *Controller* ao comando Inquiry apresentado anteriormente na tabela 4.2.

Evento	Event Code	Parametros (n bytes)
Inquiry Complete	0x01	Status (1)

Tabela 4.3: HCI Inquiry Complete Event.

Para terminar esta pequena introdução, é importante referir que a biblioteca HCI disponibilizada com a BlueZ foi bastante utilizada diretamente numa primeira fase, para tarefas como a procura de dispositivos ou a obtenção de informação relativa ao BT111. No entanto, à medida que o projeto foi sendo desenvolvido, esta abordagem foi abandonada em favor do uso da API DBus, que será apresentada em seguida.

4.2 DBus

Como já foi referido, a BlueZ disponibiliza uma API que é acessível através de um mecanismo chamado DBus, mecanismo esse que é apresentado nesta secção. O DBus não é nada mais que um mecanismo de comunicação entre processos (IPC) que são executados na mesma máquina. Também é possível usar o DBus para estabelecer comunicação entre aplicações em máquinas remotas, no entanto, não vamos abranger essa questão nesta pequena apresentação ao DBus.

Arquitetura do DBus

Basicamente, o DBus é construido à volta de um *daemon*, que é indispensável ao funcionamento deste sistema. Na realidade, é possível ter mais do que um *daemon* a serem executados independentemente na mesma máquina, formando cada um o seu próprio barramento de comunicação. Aplicações que se conectem a um dado *daemon*, passam a ser elegíveis como recetores e emissores de mensagens pelo barramento. O

funcionamento do *daemon* neste esquema será semelhante ao comportamento de um *router* numa rede de computadores, ou seja, endereçamento e encaminhamento de mensagens.

A figura 4.6 demonstra melhor este conceito.

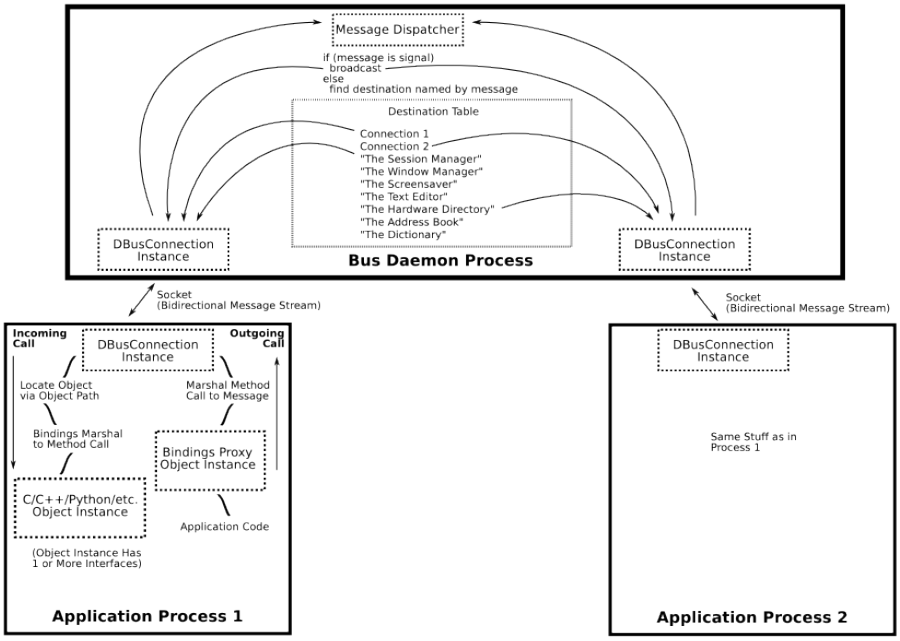


Figura 4.6: Arquitetura DBus (imagem retirada de [11]).

Na figura 4.6 estão ilustradas duas aplicações ligadas ao mesmo *daemon* DBus, que serve de intermediário. É importante salientar o conceito de *binding*, que é visível na aplicação 1. Uma *binding* não é mais que uma abstração da API de baixo nível do DBus, que os próprios programadores do DBus não aconselham a utilizar. Para isso, foram criadas as *bindings* que implementam (numa qualquer linguagem) a API de baixo nível e implementam os conceitos apresentados em seguida.

Elementos DBus

O DBus tem vários elementos [11] com diferentes funcionalidades e que é necessário conhecer de modo a perceber onde se inserem neste mecanismo e qual a sua utilidade. Vamos apresentar os conceitos por trás destes elementos para que seja mais fácil o entendimento quando se apresentarem estes conceitos em ação.

DBus Objects

Um Object é o elemento básico num processo de comunicação via DBus, na medida em que define a funcionalidade ou grupo de funcionalidades. É possível ver um Object no DBus do mesmo modo que um objeto numa linguagem de programação orientada a objetos como por exemplo Java. É basicamente o mesmo conceito.
No caso do DBus, um Object é identificado por um Object Path. Um exemplo do formato de um Object Path é o seguinte: `/org/bluez/1234/hci0`. Neste caso, este é o exemplo do Object Path dado pela BlueZ ao BT111 ligado via USB. Se for pretendido realizar operações sobre este objeto isto é possível, usando o Path indicado. Um objeto é composto por Members, que são apresentados em seguida.

DBus Members

Os Members de um Object podem ser de dois tipos:

- Methods
- Signals

Começando pelos Methods, pode-se pensar neste conceito tal e qual se pensaria em métodos C++ ou Java. Estes são operações que se podem invocar sobre o Object em questão. Os Methods podem ter parâmetros de *input* e *output*. Um exemplo de um Method, retirado da Adapter API da BlueZ:

```
void StartDiscovery()
```

Invocar este método faz com que o dispositivo Bluetooth inicie o processo de descoberta de dispositivos.

Relativamente aos Signals, estes são mensagens enviadas em *broadcast* por um Object. Aplicações que estejam à escuta do Object que originou o sinal podem receber o Signal. Os Signals podem transportar dados no caso de ser necessário passar alguma informação ao recetor para além do evento em si, no entanto não é esperado retorno pela entidade que envia o sinal. Um exemplo de um Signal, retirado da Adapter API da BlueZ:

`DeviceFound(string address, dict values)`
Este sinal é despoletado quando um dispositivo Bluetooth é encontrado num processo de descoberta de dispositivos. Cada dispositivo encontrado irá originar um sinal com a informação relativa a esse dispositivo.

DBus Interfaces

Uma Interface segue exatamente o mesmo conceito que as interfaces em C++ ou Java. No DBus, um Object pode implementar uma ou mais interfaces, que não é mais que um conjunto de Methods e Signals. Uma Interface é identificada por uma String. Um exemplo de uma Interface, neste caso a Adapter Interface da API da BlueZ: `org.bluez.Adapter`.

DBus Proxies

Para terminar, convém referir o conceito de Proxy. Basicamente, uma Proxy é uma representação local de um objeto remoto contido noutro processo. Este conceito está bastante relacionado com o conceito de RPC. Algo importante de referir é que a API de baixo nível do DBus não implementa este conceito, no entanto, a maioria dos *bindings* implementam, de maneira a tornar o trabalho dos programadores mais fácil.

Comunicação usando DBus

Tendo sido explicados os conceitos básicos do DBus, é importante perceber como se processa a comunicação entre duas aplicações usando DBus. Quando um processo se conecta a um *daemon* DBus, é-lhe atribuído um Bus Name que é único. Podemos pensar neste nome como se fosse um endereço numa rede IP. Este nome permite ao *daemon* identificar um processo no barramento, o que é essencial quando é necessária a troca de mensagens de e para um determinado processo. Durante o ciclo de vida do *daemon*, um nome que já foi atribuído a um processo, não pode voltar a ser reutilizado noutro processo.

Por convenção, os Bus Names começam com o carácter `:`, de maneira que um exemplo de um Bus Name pode ser: `:1512-3`.

Para além disto, é possível um processo pedir um nome mais facilmente identificável ao *daemon* DBus. A ideia por trás disto é exatamente a mesma que motivou o DNS. É bastante mais simples identificar um processo por uma string como `org.bluez` do que é identificá-lo por um padrão numérico como `:1512-3`.

Mensagens DBus

O DBus define um modelo de mensagens bastante simples que consiste basicamente em *header* + *body*. Estas mensagens compõem a informação que é efetivamente trocada entre processos no barramento e entre processos e o *daemon*. Existem quatro tipos de mensagem:

Method Calls
: Representa a invocação de um Method referente a um Object, pertencente a um dado processo.

Method Return
: Transporta o retorno da invocação de um Method referente a um Object.

Error Messages
: Este tipo de mensagem representa um erro causado pela invocação de um Method pertencente a um Object.

Signal Messages
: Tipo de mensagem que transporta informação relativamente a uma notificação enviada por um determinado Object.

Ou seja, uma Method Call poderá ter como retorno uma mensagem do tipo Method Return ou uma Error Message, enquanto que Signal Messages são eventos gerados assincronamente.

Todos os tipos de mensagem partilham alguma informação comum no *header* da mensagem. Resumidamente, esta informação constitui os dados necessários ao encaminhamento da mensagem, contendo também as assinaturas referentes ao tipo de dados contidos no *body* da mensagem. A estrutura da mensagem é a seguinte, tal como consta na especificação do DBus [30].

- Header

  Endianness Flag
  Octeto que identifica a *endianness* dos dados da mensagem. Se o valor for um `'l'`, é usado *Little Endian*, se o valor for `'B'`, é utilizado *Big Endian* para a codificação do *header* e *body* da mensagem.

  Message Type
  Octeto que identifica o tipo de mensagem, de acordo com os tipos apresentados anteriormente.

  Control Flags
  Octeto que contém um *bitwise* OR de bits usados para controlo do fluxo da mensagem e de ações no destinatário.

  Protocol Version
  Octeto que transporta informação relativamente à versão do protocolo da aplicação emissora.

  Body Length
  Campo de 4 bytes que identifica o tamanho do *body* em bytes.

  Message Serial
  Campo de 4 bytes que é usado para identificação da resposta a uma mensagem enviada.

  Header Fields Array
  Este *array* de zero ou mais campos pode conter informação diferente consoante o tipo de mensagem.

- Body
  - Contém os dados propriamente ditos, baseado na assinatura contida no *header*.

Como é possível observar, a estrutura da mensagem não é nada complicada. É, no entanto, necessário especificar quais os dados que é possível conter no campo Header Fields Array, na medida em que este transporta dados bastante sensíveis.

Nome	Código	Tipo	Tipo Mensagem	Descrição
Invalid	0	N/A	Não permitido	Se aparecer numa mensagem, trata-se de um erro.
Path	1	Object Path	Method Call, Signal Message	Object Path do Object para o qual se destina a mensagem ou do Object que enviou a mensagem.
Interface	2	String	Method Call, Signal Message	Interface à qual pertence o Method ou Signal. Opcional para Method Calls e obrigatório para Signal Messages.
Member	3	String	Method Call, Signal Message	Designação do Method ou Signal.
Error Name	4	String	Error Message	Designação do erro.
Reply Serial	5	UInt32	Error Message, Method Return	Message Serial da mensagem para a qual esta mensagem é resposta.
Destination	6	String	Optional	Bus Name do destinatário desta mensagem.
Sender	7	String	Optional	Bus Name do emissor desta mensagem.
Signature	8	Signature	Optional	Assinatura relativa ao *body* da mensagem. Permite a identificação dos campos presentes no *body* e do seu tamanho. Se omitido, é porque não há dados na mensagem.
Unix FDS	9	UInt32	Optional	*File descriptors* associados à mensagem. Quando omitido, é possível assumir não serem necessários.

Tabela 4.4: Informação contida no Header Fields Array de uma mensagem DBus.

Para além disto, basta apenas enviar o corpo da mensagem em si, que irá conter os dados propriamente ditos, mediante a assinatura usada. Vamos supor um exemplo apresentado acima do Signal `Device Found` que tem como argumentos uma *string* e um dicionário. Portanto, no campo Signature seria observável o seguinte valor:

`sa{ss}`,

onde '`s`' identifica uma String e '`a{ss}`' identifica um dicionário com chave e valor do tipo *string*.

Tendo este caso, no *body* apenas teriam de ser incluídos os dados. De referir que o final de uma String (para efeitos de separação de parâmetros) é identificado pelo carácter '`\0`'.

Para terminar esta secção, é necessário fazer uma pequena apresentação à API disponibilizada pela BlueZ, na medida em que alguns dos Methods e Signals foram essenciais no projeto *CareStore*. Naturalmente, é impraticável fazer uma apresentação completa à API, mas são apresentados os elementos mais importantes das classes da API mais relevantes para o projeto. Mais uma vez, é crucial salientar que a API apresentada refere-se à versão 4 da BlueZ.

Apresenta-se então a tabela 4.5, que apresenta os principais elementos da API da BlueZ.

Classe	Tipo Member	Designação	Descrição
Adapter	Method	`void StartDiscovery()`	Inicia o processo de descoberta de dispositivos Bluetooth.
		`void StopDiscovery()`	Interrompe um processo de descoberta de dispositivos Bluetooth.
		`object CreatePairedDevice( string address, object agent, string capability)`	Cria o Object Path de um determinado dispositivo Bluetooth e inicia o processo de emparelhamento com esse mesmo dispositivo.
		`void RegisterAgent( object agent, string capability)`	Regista um agente de emparelhamento que irá lidar com os processos de emparelhamento de dispositivos Bluetooth.
	Signal	`DeviceFound( string address, dict values)`	Este sinal é enviado cada vez que um dispositivo Bluetooth é encontrado durante um processo de descoberta.
Agent	Method	`string RequestPinCode( object device)`	Este método é invocado quando o *daemon* necessita de uma chave para autenticação. O valor deverá ser uma String alfanumérica de 1-16 caracteres.
		`void DisplayPinCode( object device, string pincode)`	Este método é invocado quando o *daemon* necessita de exibir uma chave de autenticação.
Attribute	—	—	Esta classe é usada para aceder aos registos Service Discovery Protocol de um dispositivo Bluetooth.
Audio	—	—	Utilizada para funcionalidades relacionadas com áudio.
Control	—	—	Utilizada para tarefas de controlo sobre um dispositivo remoto.

Classe	Tipo Member	Designação	Descrição
Device	Method	`dict DiscoverServices( string pattern)`	Permite a descoberta de serviços disponibilizados pelo dispositivo sobre o qual o método é invocado.
		`dict GetProperties()`	Permite a descoberta das propriedades do dispositivo sobre o qual o método é invocado. Algumas propriedades interessantes são o endereço do dispositivo, a sua classe de dispositivo ou a lista de serviços que disponibiliza.
Health	Method	`object CreateApplication( dict config)`	Interage com a implementação da BlueZ do Health Device Profile de maneira a criar uma aplicação para recolha ou envio de dados médicos.
		`object CreateChannel( object application, string configuration)`	Cria um novo canal de dados de modo a trocar informação médica.
	Signal	`void ChannelConnected( object channel)`	Este sinal notifica a aplicação que o recebe da criação ou reconexão de um canal de dados para troca de informação médica.
HFP	—	—	Esta classe é usada para dar suporte a funcionalidades mãos livres.
Input	—	—	Esta classe permite o estabelecimento de conexões com dispositivos de entrada como teclados ou ratos.

Classe	Tipo Member	Designação	Descrição
Manager	Method	`object DefaultAdapter()`	Este método permite obter o Object Path do dispositivo Bluetooth local.
		`object FindAdapter( string pattern)`	Permite obter o Object Path de um dispositivo Bluetooth local específico. Isto é especialmente útil se quisermos identificar um dispositivo específico ou se tivermos mais de um dispositivo conectado.
	Signal	`AdapterAdded( object adapter)`	Este sinal indica o recetor de que um novo dispositivo Bluetooth foi conectado localmente.
		`DefaultAdapterChanged( object adapter)`	Quando existe mais que um dispositivo conectado localmente (por exemplo, dois *dongle*'s Bluetooth), um dos dispositivos é escolhido como o dispositivo por defeito. Quando o dispositivo por defeito é alterado, este sinal notifica qual é o novo dispositivo.
Media	—	—	Garante funcionalidades relacionadas com multimédia.
Network	—	—	Garante funcionalidades relacionadas com *networking*.
SAP	—	—	Garante funcionalidades relacionadas com o SIM Card de dispositivos móveis.
Serial	—	—	Garante funcionalidades relacionadas com comunicações RFCOMM com dispositivos remotos.
Service	—	—	Permite a configuração de serviços no dispositivo Bluetooth local.

Tabela 4.5: API DBus da BlueZ.

Capítulo 5

Implementação do Sistema

Agora que estão apresentadas as tecnologias envolvidas no projeto *CareStore*, é finalmente possível partir para a implementação efetiva do sistema. Neste capítulo será explicado em detalhe o que foi implementado nos protótipos desenvolvidos. Como já foi referido anteriormente, este projeto não foi desenvolvido por uma só pessoa, tratando-se de um esforço conjunto de várias pessoas, representando entidades diferentes.

Indo de encontro ao tema desta dissertação, serão apresentados os detalhes de implementação relativamente aquelas que foram as tarefas do autor desta dissertação no projeto *CareStore*, ou seja, funcionalidades Bluetooth.

- Implementação de todas as tarefas relacionadas com tecnologias Bluetooth;
  - Procura de dispositivos;
  - Deteção de dispositivos previamente emparelhados;
  - Emparelhamento de dispositivos;
  - Comunicação com dispositivos médicos emparelhados;
- Implementação de mecanismo de emparelhamento Bluetooth via NFC;
- Preparação da imagem do SO de suporte à plataforma CRIP.

Esta listagem foi simplificada de modo a que se consiga perceber bem as tarefas a desenvolver. Como se irá perceber em seguida, as ferramentas utilizadas para levar a

cabo essas tarefas, tornaram o desenvolvimento um processo algo não-linear, devido à sua complexidade.

Algo importante de referir é que ao longo do desenvolvimento do projeto, os próprios objetivos do projeto foram sendo alterados, e por vezes foi necessário refazer ou adaptar uma componente previamente considerada concluída. Estas questões serão explicadas ao longo deste capítulo, quando assim for pertinente, de maneira a que mais facilmente se perceba a evolução do projeto.

Deve também referir-se como foi crucial, para a implementação do projeto, o agendamento de reuniões regulares entre as entidades envolvidas no projeto, facto que já foi referido anteriormente. Isto possibilitou a troca de ideias relativamente ao rumo que o projeto deveria tomar e permitiu maior celeridade em todo o processo de implementação. Tendo em conta que eventualmente foi necessária a integração entre os sistemas desenvolvidos pela Universidade de Aarhus (CAALHP) e a Universidade do Minho (CRIP), a realização de reuniões presenciais e a troca constante de documentação para especificação da comunicação entre os dois sistemas foi absolutamente essencial para o desenvolvimento do projeto.

5.1 Funcionalidades Bluetooth no CareStore

Feita uma introdução mais técnica a algumas das ferramentas utilizadas no projeto *CareStore*, resta explicar a implementação das funcionalidades em si. No início deste capítulo, foram apresentados os objetivos de implementação do projeto, no que toca a funcionalidades Bluetooth, pelo que será explicada a implementação de cada um individualmente. Como já foi referido em alguns pontos neste documento, a implementação do projeto *CareStore* foi gradual, no sentido em que foram desenvolvidos vários protótipos, até se atingir a versão atual do sistema CRIP. Para cada um dos objetivos definidos, poderão ser apresentadas soluções entretanto substituídas por soluções mais eficientes.

5.1.1 Descoberta de dispositivos Bluetooth

A tarefa de procura de dispositivos Bluetooth é transversal a todas as versões desta tecnologia e, portanto, essencial para o funcionamento de qualquer sistema que a queira usar. No *CareStore* é essencial a procura de dispositivos médicos com capacidades Bluetooth, no sentido de os emparelhar, habilitando-os a comunicar com o CRIP.

O desenvolvimento desta funcionalidade passou por duas fases:

- No primeiro protótipo, entregue em Dezembro de 2013, a descoberta de dispositivos era feita usando a API HCI, disponibilizada pela *stack* BlueZ;
- A partir do segundo protótipo, entregue em Maio de 2014, esta funcionalidade é implementada usando a API DBus, disponibilizada pela *stack* BlueZ.

Ambas as implementações serão apresentadas individualmente e no fim será feita a comparação entre as duas, de modo a perceber as razões pelas quais foi necessária a mudança de estratégia.

Descoberta via HCI

Na altura em que o primeiro protótipo foi desenvolvido, necessitávamos apenas de duas informações: o Nome do dispositivo e o seu Endereço único. Apenas a primeira

é retornada no processo de descoberta propriamente dito, sendo que para saber o nome do dispositivo é necessário o estabelecimento de conexão com cada dispositivo individualmente. Para o desenvolvimento desta funcionalidade, foi utilizada a API disponibilizada pela BlueZ para este efeito.

Foi desenvolvido o algoritmo apresentado na figura 5.1.

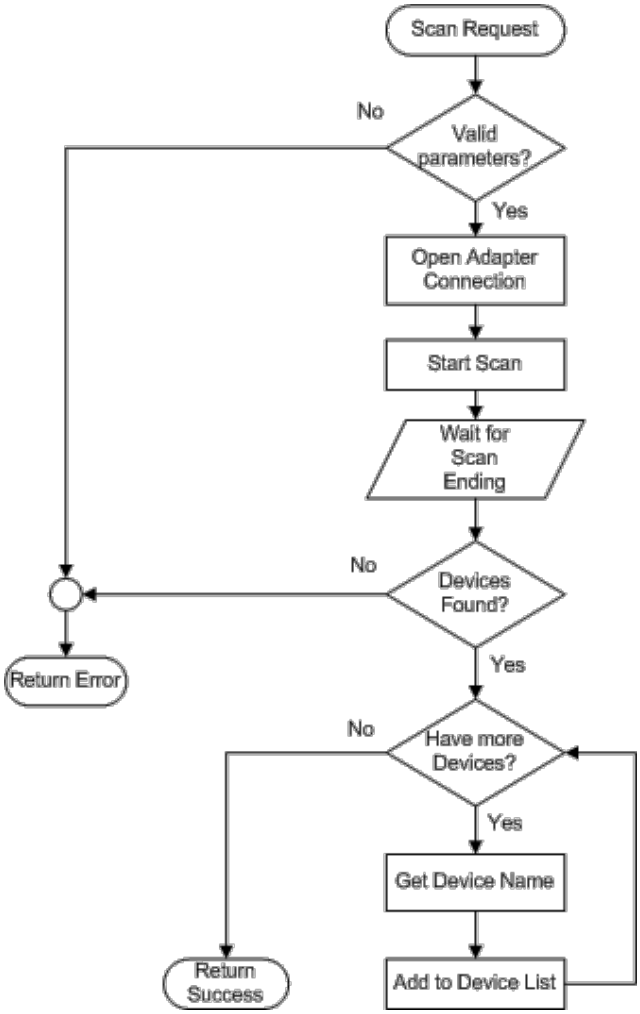


Figura 5.1: Algoritmo de descoberta de dispositivos Bluetooth do 1º protótipo.

O mais importante a notar neste método é que a descoberta do endereço único do dispositivo e do nome do dispositivo são realizados separadamente, dado que é necessário conhecer o endereço do dispositivo para saber o seu nome. Na figura 5.1, é possível observar que primeiro há um passo de descoberta de dispositivos, que irá

retornar o endereço dos mesmos, sendo que, posteriormente, essa lista é percorrida de modo a descobrir os nomes dos dispositivos. Esta abordagem foi abandonada em favor da apresentada em seguida.

Descoberta via DBus

O segundo método de descoberta de dispositivos Bluetooth trata-se de um método mais recente que é disponibilizado pela Adapter DBus API, disponibilizada pela BlueZ. Este método utiliza o mecanismo de EIR [17] que nos permite obter bastante mais informação acerca dos dispositivos encontrados. Nomeando alguns dos mais importantes:

- Nome;
- Endereço;
- Classe;
- Serviços disponibilizados;
- RSSI;
- Tipo de emparelhamento suportado.

É interessante conhecer algumas destas propriedades antes de sequer emparelhar o dispositivo. Por exemplo, supondo que o sistema apenas irá suportar dispositivos médicos, através da classe de dispositivo é possível fazer essa filtragem. Também é possível apenas querer emparelhar dispositivos que forneçam um serviço muito específico, pelo que seria interessante analisar a lista de serviços disponibilizados para garantir que o serviço em questão é implementado no dispositivo. No caso do projeto *CareStore* não foi necessário haver restrições a este nível.

Para terminar, na figura 5.2 apresenta-se o algoritmo que foi implementado para dar suporte a esta funcionalidade.

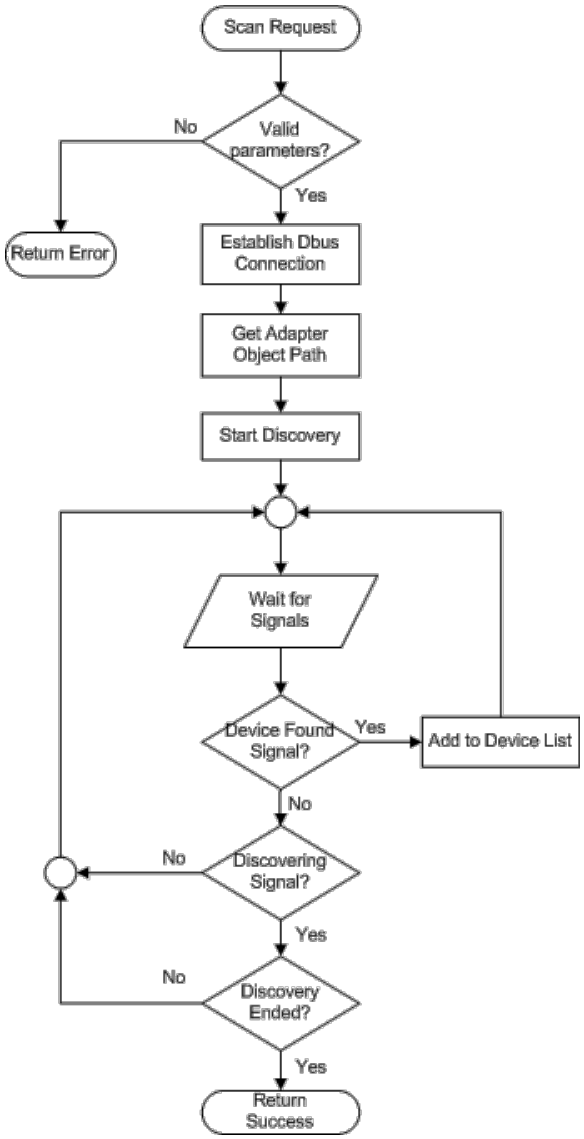


Figura 5.2: Algoritmo de descoberta de dispositivos Bluetooth a partir do 2º protótipo, utilizando DBus.

Este método de descoberta é o que se mantém em funcionamento no projeto *CareStore*, dado se ter revelado bastante eficiente.

O método DBus utilizado para iniciar um processo de descoberta faz parte da Adapter DBus Interface da BlueZ e deve ser invocado do seguinte modo:

```
void StartDiscovery()
```

Este método irá fazer com que o BT111 inicie a descoberta de dispositivos Bluetooth nas imediações. A descoberta de um dispositivo origina um sinal `DeviceFound` que transporta dados originários do processo de Extended Inquiry Response, referentes ao dispositivo Bluetooth encontrado. Deste modo é possível saber bastante informação acerca do dispositivo em questão.

Para terminar esta secção, é importante perceber o porquê da mudança entre os dois métodos apresentados. Aquando da entrega do primeiro protótipo, foram notados alguns problemas no processo de descoberta usando HCI. O processo de descoberta propriamente dito apresentava fiabilidade, no entanto, a descoberta do nome do dispositivo poderia revelar-se demorada devido a dois fatores essenciais.

A primeira razão é o facto de ser necessário o estabelecimento de uma conexão com cada um dos dispositivos descobertos de modo a inquirir o nome do dispositivo. Tendo em conta que é possível haver bastantes dispositivos Bluetooth nas imediações, facilmente se compreende como esta estratégia se afigura impraticável.

A segunda razão prende-se com o facto do processo de comunicação com o dispositivo, com vista à descoberta do seu nome, se revelar um processo bastante demorado. Segundo alguns testes desenvolvidos, este problema era intensificado num ambiente onde a banda ISM de 2,4GHz era amplamente utilizada. Por exemplo, foram desenvolvidos testes numa sala onde essa banda de frequências se apresentava bastante congestionada, devido à existência de um elevado número de aparelhos a estabelecer comunicação via WiFi e 802.15.4 e, nessas condições, a descoberta apresentava-se demorada, sendo que, por vezes, não era possível a extração do nome do dispositivo. Se o meio se revelasse pouco congestionado, a extração do nome decorria rapidamente e sem erros.

Pelas razões explicadas, rapidamente se percebeu que esta estratégia não era eficiente e em casos extremos, era simplesmente impraticável. De modo que após algum estudo foi descoberto o EIR, cujo desempenho melhora bastante, mesmo em ambientes ruidosos. Este método, disponível através da DBus API, permite o acesso a bastante mais informação através do processo de descoberta propriamente dito, não sendo necessário o estabelecimento de qualquer conexão direta com os dispositivos.

5.1.2 Deteção de dispositivos Bluetooth

No projeto *CareStore* a mobilidade dos dispositivos médicos é um dos objetivos, por isso, torna-se importante existir um método que permita saber quais os dispositivos emparelhados com o CRIP que se encontram ao alcance do mesmo. Ou seja, não é feito um *scan* à rede, mas sim uma procura dos dispositivos habilitados a interagir com o sistema do *CareStore*, na medida em que se tenta estabelecer uma conexão com os dispositivos emparelhados com o sistema. Mais uma vez, a versão apresentada no primeiro protótipo não é a que figura nas versões mais recentes, por razões que serão explicadas ao longo desta secção.

Algo de referir é que foi usada a DBus API em ambas as versões, de maneira a descobrir quais os dispositivos emparelhados com o CRIP. A diferença entre as estratégias assenta no método usado para descobrir se estes dispositivos se encontravam nas imediações.

Antes de entrar em detalhe relativamente a este assunto, deve ser explicado como foi obtida a lista de dispositivos emparelhados, visto este aspeto da implementação ser igual em todos os protótipos.

Para isto é usado um método da classe Adapter da API DBus da BlueZ:

```
dict GetProperties()
```

Este método devolve as propriedades do Adapter Bluetooth, entre as quais se inclui uma lista dos dispositivos com os quais se emparelhou sob a forma de um *array* de Object Paths referentes aos objetos DBus dos dispositivos emparelhados.

Deteção no primeiro protótipo

No primeiro protótipo, foi utilizada uma estratégia bastante simples. Utilizou-se o servidor SDP existente em todos os aparelhos que implementam a *stack* Bluetooth. Se for possível estabelecer uma conexão com o servidor SDP de um dado dispositivo, naturalmente o dispositivo estará nas imediações.

O algoritmo implementado é apresentado na figura 5.3.

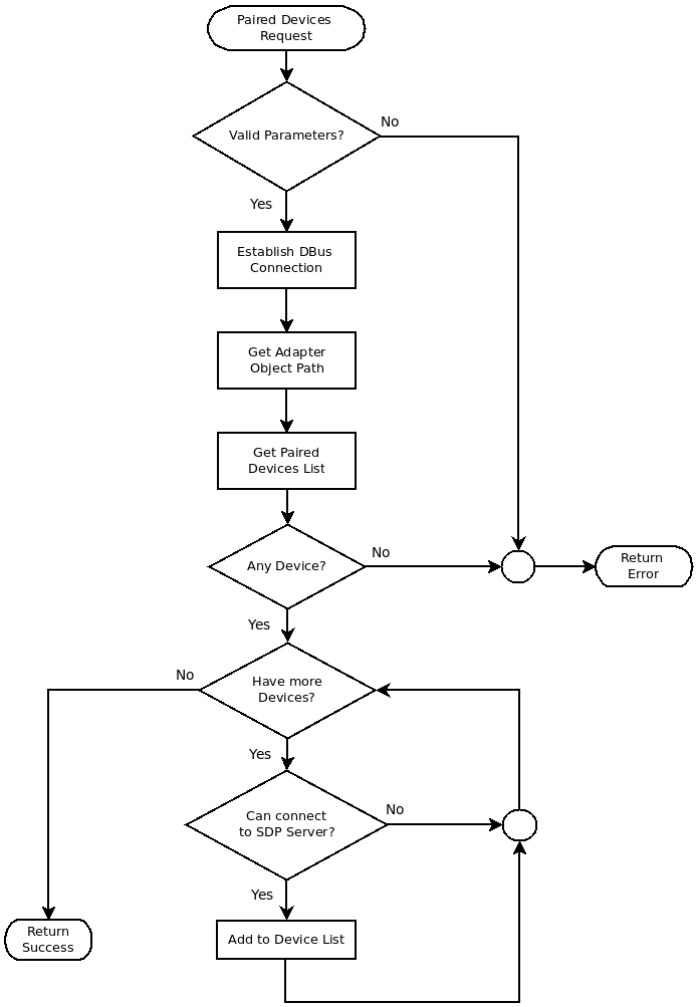


Figura 5.3: Algoritmo de deteção de dispositivos Bluetooth no 1º protótipo.

Este método apresenta os resultados esperados, ou seja, retorna uma lista com os dispositivos emparelhados com o CRIP e com os quais foi possível estabelecer comunicação.

Deteção a partir do segundo protótipo

Enquanto que a implementação anterior não apresentou nada de errado em termos funcionais, é incorreto usar o servidor SDP, que está destinado a descobrir os serviços fornecidos pelo dispositivo, para uma tarefa de verificação da presença do dispositivo nas imediações do CRIP. Não é com esta finalidade que o servidor SDP deve ser utilizado, sendo necessário repensar como verificar a presença de uma dado

dispositivo.

Isto foi resolvido usando a camada do Logical Link Control and Adaptation Protocol (L2CAP), que recorre a conexões Asynchronous Connection-Less (ACL) de modo a estabelecer uma conexão genérica entre dispositivos Bluetooth. Deste modo não é necessário recorrer ao servidor SDP, sobrecarregando-o sem razão, visto que não pretendemos saber quais os serviços disponibilizados pelo dispositivo.

O algoritmo adotado, apresentado na figura 5.4, mantém-se semelhante ao apresentado na figura 5.3, diferenciando-se apenas na componente utilizada para deteção do dispositivo.

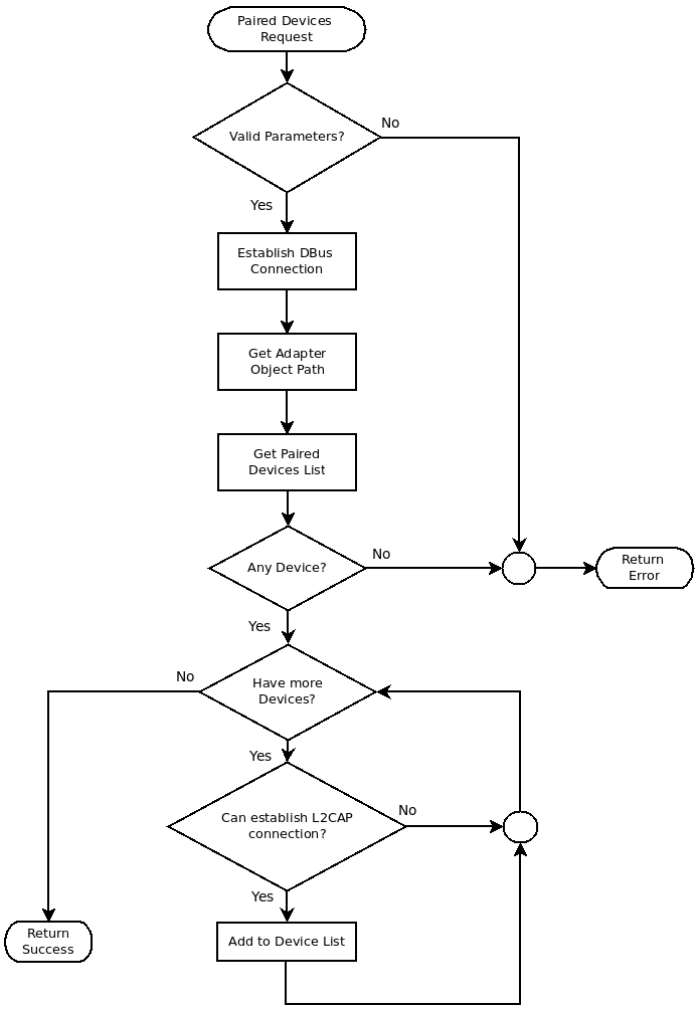


Figura 5.4: Algoritmo de deteção de dispositivos Bluetooth a partir do 2º protótipo.

Para terminar, é importante referir que os dispositivos poderão estar dotados de um modo que os possibilite adormecer, fazendo com que não possam aceitar conexões Bluetooth. Isto é um esquema implementado pelos fabricantes, tendo em conta que muitos destes dispositivos têm restrições em termos energéticos (por exemplo, muitos dos dispositivos médicos utilizados são alimentados por pilhas). Portanto, se o módulo Bluetooth de um dispositivo é desligado, naturalmente que ele não será detetado utilizando este método, visto que não tem capacidade de comunicação.

5.1.3 Emparelhamento de dispositivos Bluetooth

Este é provavelmente o processo mais característico e que mais facilmente se associa às tecnologias Bluetooth. O emparelhamento de dispositivos é essencial dado que permite que dois dispositivos se conectem e, potencialmente, acedam aos serviços disponibilizados por ambos. Este passo é necessário, visto que alguns desses serviços poderão envolver a troca de dados privados e é importante para o utilizador controlar quais os dispositivos que poderão conectar-se a si. Para além disto, pode ser interessante, para alguns serviços, ter a capacidade de estabelecer uma conexão sem interação com o utilizador (Por exemplo, ao entrar no carro, o telemóvel automaticamente começar a tocar usando o sistema de som do automóvel).

No caso do *CareStore* este foi um dos processos que mais alterações sofreu de protótipo para protótipo, à medida em que os requisitos de projeto foram mudando:

- No primeiro protótipo, entregue em Dezembro de 2013, o emparelhamento de dispositivos era assegurado por ferramentas externas, não integradas no código do projeto *CareStore*, visto o método como era feito o emparelhamento de dispositivos não ser uma prioridade;
- No segundo protótipo, entregue em Maio de 2014, o emparelhamento de dispositivos era assegurado por ferramentas externas, integradas no código do projeto *CareStore*. Nesta fase, já era um requisito a API do *CareStore* suportar o emparelhamento de dispositivos. Para além disto, foi dado suporte a um esquema de emparelhamento *Out of Band* usando NFC;
- No terceiro e atual protótipo, apresentado em Outubro de 2014, o emparelhamento de dispositivos passou a ser assegurado por um agente de emparelha-

mento desenvolvido especialmente tendo em vista as necessidades do projeto *CareStore*. Deste modo, evitou-se a dependência de ferramentas externas.

É importante analisar todo o processo de desenvolvimento desde o esquema apresentado no primeiro protótipo até aquela que é a solução atual.

Relativamente ao primeiro esquema de emparelhamento, não há muito a explicar, visto que este era assegurado por ferramentas externas, incluídas no pacote *bluez-tools*. Nesta altura, o foco do projeto era assegurar a extração de dados médicos dos dispositivos, de modo que, o método pelo qual estes dispositivos eram emparelhados não era muito importante, desde que esse emparelhamento fosse garantido. Ou seja, o que era feito era emparelhar os dispositivos com recurso a uma ferramenta externa (*bt-device*).

Naturalmente, num caso real, seria essencial que essa funcionalidade estivesse integrada no sistema CRIP, dado que os equipamentos estão em constante mobilidade e não temos o pressuposto de que os equipamentos estão emparelhados com o CRIP. Para além disto, o projeto *CareStore* apresenta características bastante peculiares que exigem que o emparelhamento seja feito de modo bastante cuidado.

Para o segundo protótipo, um dos requisitos seria que o emparelhamento de dispositivos fizesse parte da API do CRIP. Para além disto, surgiu a ideia de tornar o processo bastante mais simples, evitando que os utilizadores do sistema tivessem de ter qualquer conhecimento tecnológico para emparelhar um dispositivo com o sistema. Num processo de emparelhamento normal, existe um primeiro passo de descoberta de dispositivos, seguido do emparelhamento propriamente dito, que geralmente envolve um PIN que tem de ser inserido pelo utilizador. No *CareStore* pretendeu-se tornar todo este processo o mais célere e transparente possível. Tendo isto em vista, foi desenvolvido um esquema de emparelhamento *Out of Band* (OOB), baseado em NFC. Será reservada uma secção para explicar o trabalho desenvolvido relativamente à componente NFC deste modelo de emparelhamento. Neste momento, é pretendido cingir a informação a processos exclusivamente Bluetooth.

Como já foi referido, em termos de processo Bluetooth propriamente dito, o processo manteve-se bastante inalterado relativamente ao primeiro protótipo. O emparelhamento era assegurado pela aplicação `bt-device`, sendo que esta passou a ser

integrada no código do projeto. Apesar de se manter a possibilidade de realizar a descoberta de dispositivos, usando o modelo de emparelhamento baseado em NFC, deixa de ser necessário este passo, na medida em que a informação necessária ao emparelhamento de dispositivos é obtida via processos de comunicação NFC.

O algoritmo implementado é apresentado na figura 5.5.

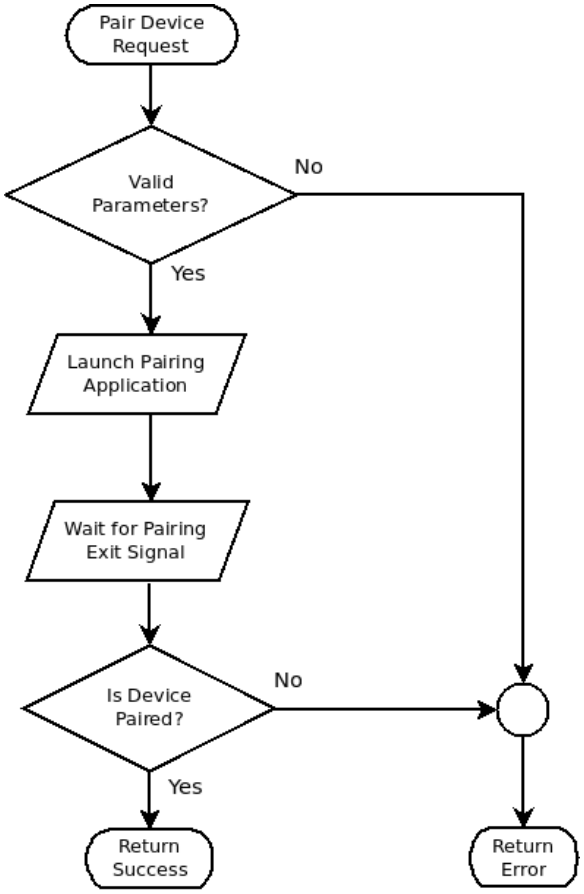


Figura 5.5: Algoritmo de emparelhamento de dispositivos Bluetooth no 2º protótipo.

Naturalmente, esta ainda não se trata de uma solução ideal, visto ser dependente de uma ferramenta externa, cujo fluxo de desenvolvimento é completamente independente ao *CareStore*. Para além disto, a aplicação não tinha em conta as necessidades peculiares encontradas no *CareStore*, de modo que foi sempre o objetivo final desenvolver um agente de emparelhamento sobre o qual fosse possível ter completo controlo e feito à medida do projeto.

Como já foi referido, com vista à preparação do terceiro protótipo, o principal foco do projeto tratou-se do desenvolvimento de um agente de emparelhamento adequado às características do projeto *CareStore*. O modelo de emparelhamento final teria de ir de encontro aos seguintes objetivos:

- Evitar, no uso do sistema *CareStore*, o processo de descoberta de dispositivos Bluetooth, visto que esse processo, para além de ser demorado, é geralmente despoletado por um utilizador;
- Garantir que não é necessário ao utilizador ter de inserir nenhum tipo de código alfanumérico, em nenhuma fase do processo de emparelhamento.

Como foi referido, um dos objetivos do *CareStore* é a transparência e facilidade de uso. O primeiro objetivo já havia sido atingido no segundo protótipo, visto ter sido desenvolvido suporte a toda a componente NFC do processo. O segundo objetivo apenas foi assegurado com o desenvolvimento de raiz de um agente de emparelhamento adequado ao projeto.

Antes de avançar, é importante perceber que as capacidades de entrada e saída do CRIP são inexistentes. O CRIP não está equipado de nenhum teclado para operações de entrada, nem possui nenhum ecrã onde seja possível apresentar *feedback* ao utilizador.

A especificação Bluetooth define uma lista com as capacidades IO de dispositivos Bluetooth. Esta informação é usada no processo de emparelhamento, de modo aos dois dispositivos envolvidos no processo poderem dar a conhecer as suas capacidades individuais de IO ao dispositivo com o qual se tentam emparelhar. Isto permite com que seja possível a adaptação do processo de emparelhamento, de modo a ir de encontro às especificações dos dois dispositivos.

A lista, apresentada na tabela 5.1, define o conjunto de capacidades de IO, tal como definidas na especificação Bluetooth.

Capacidade IO	Descrição
DisplayOnly	O dispositivo apenas permite apresentar informação, não tendo nenhum mecanismo de *input*.
DisplayYesNo	O dispositivo permite apresentar informação e possui um mecanismo de modo a pedir confirmação da operação de emparelhamento ao utilizador.
KeyboardOnly	O dispositivo não tem capacidades de *output*, mas possui um teclado de modo inserir informação.
NoInputNoOutput	O dispositivo não tem quaisquer mecanismos de IO.

Tabela 5.1: Capacidades IO de dispositivos Bluetooth [17].

Analisando a tabela 5.1, facilmente se percebe que o CRIP se insere na categoria `NoInputNoOutput`. Isto irá fazer com que os dispositivos que se tentam emparelhar com o CRIP tenham de se adaptar ao facto deste não ter qualquer mecanismo de IO.

Até à versão 2.1 da especificação Bluetooth, era obrigatória a introdução de um PIN numérico, igual em ambos os dispositivos, de modo a confirmar o processo de emparelhamento, forçando a interação com o utilizador. Se um dos dispositivos, por exemplo, um *headset*, não possuía capacidade de *input*, o PIN era *hardcoded* no dispositivo. Atualmente, este processo ainda é suportado de modo a não descontinuar dispositivos mais antigos, mas é referido como Legacy Pairing.

A partir desta versão, foi introduzido o conceito de Secure Simple Pairing (SSP), que permite mecanismos de associação onde não é necessária a interação com o utilizador (*Just Works*), apesar de manter o suporte a esquemas onde pode ser usada uma *passkey* alfanumérica, se assim for desejado. Para além disto, o SSP é bastante mais seguro que os esquemas de Legacy Pairing, graças ao uso de criptografia de chave pública e mecanismos de proteção contra ataques MITM. Para o projeto *CareStore* isto é extremamente interessante, indo de encontro ao objetivo de retirar carga ao utilizador, com o bónus de tornar o processo mais seguro.

De modo a implementar um agente de emparelhamento, a BlueZ obriga a implementar a interface Agent, pertencente à classe Agent da API DBus da BlueZ. Implementado um agente, este pode ser registado na BlueZ, passando a ser utilizado

nos processos de emparelhamento futuros. De modo a registar um agente de emparelhamento, terá de ser usada a classe Adapter da API DBus. Esta classe também é usada para iniciar o emparelhamento com um dispositivo Bluetooth remoto.

Visto ser necessária a implementação de um agente, isso implica a criação de um objeto DBus (o seu Object Path pode ser definido livremente), sobre o qual podem ser invocados os métodos DBus definidos na interface Agent. A invocação desses métodos provém do *daemon* Bluetooth disponibilizado com a BlueZ. Os métodos são os seguintes:

- `void Release()`
- `string RequestPinCode(object device)`
- `uint32 RequestPasskey(object device)`
- `void DisplayPasskey(object device, uint32 passkey, uint8 entered)`
- `void DisplayPinCode(object device, string pincode)`
- `void RequestConfirmation(object device, uint32 passkey)`
- `void Authorize(object device, string uuid)`
- `void ConfirmModeChange(string mode)`
- `void Cancel()`

No caso *CareStore*, visto o CRIP não ter mecanismos de IO, alguns destes métodos não são relevantes. Se, por exemplo, o CRIP fosse disponibilizado com um ecrã, os métodos de `Display` poderiam ser programados do modo que fosse mais relevante para exibir a informação. Os métodos `Display` e `Request` nunca serão invocados no CRIP, visto que este é registado como tendo capacidades *NoInputNoOutput*.

Relativamente aos outros métodos, o método `Release` é invocado pelo *daemon* Bluetooth quando o agente é removido, de modo a realizar operações de limpeza de memória ao nível da aplicação do agente. No caso do *CareStore*, não existe nenhuma operação a realizar no caso do agente ser removido.

O método `Authorize` é invocado quando o *daemon* Bluetooth necessita de autorização para aceitar a conexão de um dispositivo. O Object Path DBus é indicado como parâmetro do método, pelo que se torna fácil descobrir informações relativas ao

dispositivo em questão, através da classe Device da API DBus, se assim for pretendido. No *CareStore*, não foram especificadas restrições quanto ao tipo de dispositivos que se podem conectar ao CRIP, de modo que não existem restrições a este nível.

O método `Cancel` é utilizado quando é pretendido cancelar uma invocação sobre o agente. Por exemplo, se fosse invocado o método `DisplayPasskey`, seria usado este método para fazer com que a *passkey* desaparecesse do ecrã. Por isso, este método não é muito relevante para o *CareStore*.

Sendo implementado o agente, é importante registá-lo, de modo a que o *daemon* Bluetooth possa invocar os métodos referidos acima e avançar o processo de emparelhamento. Para isto, é utilizado um método da classe Adapter da API DBus:

- `void RegisterAgent(object agent, string capability)`

Este método permite registar um agente que irá auxiliar o processo de emparelhamento, tal como explicado anteriormente. No caso do CRIP, o agente é registado como tendo as capacidades *NoInputNoOutput*. Relativamente ao primeiro parâmetro, terá de ser passado o Object Path definido para o agente. Indo de encontro àquilo que já foi explicado acerca dos mecanismos DBus, o agente do CRIP é registado com o seguinte Object Path: `/esrg/carestore`. A partir do momento em que o agente é registado no *daemon* Bluetooth, outras aplicações podem fazer invocações aos métodos do agente de emparelhamento, sendo que este redireciona as mensagens para o Object identificado com o Object Path `/esrg/carestore`.

Um agente pode ser removido a qualquer momento, recorrendo ao seguinte método da classe Adapter da API DBus:

- `void UnregisterAgent(object agent)`

Explicado como foi desenvolvido o agente, basta apenas referir como utilizar a API da BlueZ para efetuar um pedido de emparelhamento com um dispositivo Bluetooth. Para isto, é necessário recorrer mais uma vez à classe Adapter da API DBus, que oferece um método para efetuar o emparelhamento com dispositivos Bluetooth:

- `object CreatePairedDevice(string address, object agent, string capability)`

Este método necessita apenas do endereço do dispositivo Bluetooth com o qual é pretendido realizar o emparelhamento. Relativamente aos outros dois parâmetros, são equivalentes aos usados no método `RegisterAgent`, sendo que, neste caso, permitem que se defina um agente para ser usado nesta instância particular de emparelhamento. Ou seja, é possível definir individualmente o agente que irá realizar a gestão de um processo de emparelhamento.

No caso do projeto *CareStore*, na sua versão atual, é sempre o CRIP que inicia os processos de emparelhamento, e utiliza o método apresentado. Quanto ao endereço do dispositivo, como já foi referido, este é obtido através de um mecanismo NFC, que será explicado em seguida.

O algoritmo implementado para proceder ao emparelhamento de um dispositivo Bluetooth no terceiro protótipo do projeto *CareStore* é então o apresentado na figura 5.6.

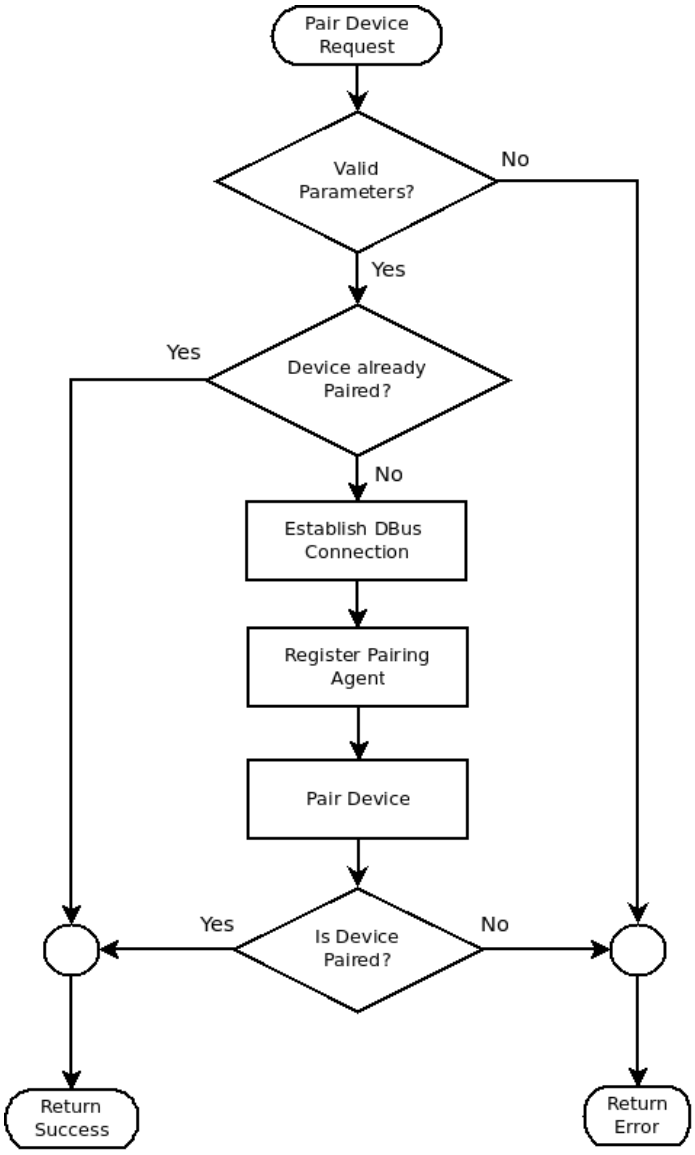


Figura 5.6: Algoritmo de emparelhamento de dispositivos Bluetooth no 3º protótipo.

Mais uma vez, este mecanismo apenas cobre a componente Bluetooth do processo, sendo que a componente OOB é explicada em seguida.

5.1.3.1 Emparelhamento OOB via NFC

Tal como referido, este mecanismo de emparelhamento tem algumas características que são bastante interessantes para projeto *CareStore*. Relembrando:

- O processo torna-se bastante transparente para o utilizador. Do ponto de vista do utilizador (provavelmente um profissional de saúde), basta passar uma *tag* NFC por um leitor e o dispositivo médico é automaticamente emparelhado;
- O processo é bastante mais rápido que o habitual esquema de descoberta seguido de seleção de dispositivo e emparelhamento. Usando NFC, deixa de ser necessário o passo de descoberta, sendo que a informação necessária ao emparelhamento está contida numa *tag* NFC;
- Para ser estabelecida comunicação NFC, é necessária proximidade entre os dois dispositivos (<10cm), pelo que isto também contribui para a segurança do sistema.

Estas razões são mais que suficientes para perceber a utilidade deste mecanismo de emparelhamento, num projeto com as características do *CareStore*. É importante referir que o NFC não é a única tecnologia *Out of Band* que pode ser utilizada, no entanto, esta já estava a ser utilizada no projeto para a identificação de utilizadores registados com a plataforma CRIP, por isso faz todo o sentido utilizar um recurso que já estava disponível para esta finalidade. Outras tecnologias *Out of Band* que podem ser utilizadas incluem WiFi ou USB, entre outras. Antes de avançar, é importante salientar que o processo de emparelhamento em si é um processo exclusivamente Bluetooth. O NFC é utilizado como um mecanismo para obter toda a informação necessária de modo a despoletar o processo de emparelhamento, sendo que, obtida essa informação, é realizado o *handover* para um processo inteiramente Bluetooth.

A implementação descrita nesta secção segue as especificações definidas pelo NFC Forum, relativamente ao emparelhamento *Out of Band* usando NFC, de modo a assegurar a compatibilidade com soluções existentes no mercado.

Tendo isto, o NFC Forum define dois tipos de *handover* [12].

Negotiated Handover

Quando temos dois dispositivos, dotados de capacidades de comunicação NFC ativa, com várias tecnologias de comunicação disponíveis, esses dois dispositivos podem negociar qual a tecnologia para a qual vão fazer *handover*. Um dos dispositivos. o *Requester*, indica as tecnologias de comunicação que tem disponíveis e envia a informação ao outro dispositivo, o *Selector*, que faz o mesmo. O *Requester* pode então escolher a tecnologia que mais se adapta, de modo a realizar a operação de *handover*:

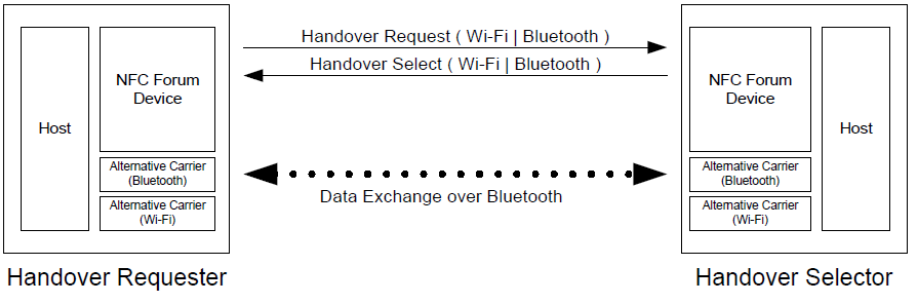


Figura 5.7: Mecanismo de Negotiated Handover (imagem retirada de [12]).

No caso do *CareStore*, o CRIP, que teria o papel de *Requester*, apenas indicaria Bluetooth como tecnologia disponível. No entanto, esta solução não é muito relevante no estado atual do projeto *CareStore*, visto que os dispositivos médicos não tem capacidades de comunicação NFC. Se imaginarmos que o sistema poderia comunicar com um *smartphone*, dotado de capacidade de comunicar via NFC, este já seria um esquema interessante para, por exemplo, emparelhar o CRIP com esse *smartphone*.

O segundo tipo de *handover* é bastante mais interessante para o projeto *CareStore*.

Static Handover

Neste caso, quando um dos dispositivos não suporta comunicação NFC, podemos dotá-lo de uma *tag* NFC, que poderá ser escrita com dados estáticos. Essa *tag* seria escrita com os dados referentes às tecnologias de comunicação suportadas pelo dispositivo, de modo simular a resposta de um *Selector*.

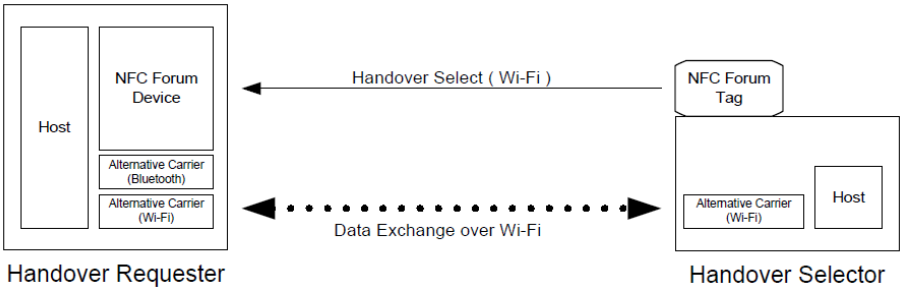


Figura 5.8: Mecanismo de Static Handover (imagem retirada de [12]).

No caso do *CareStore*, os dispositivos médicos seriam equipados com uma *tag* onde seriam escritos os dados necessários ao emparelhamento com o CRIP. Como os dispositivos médicos apenas possuem Bluetooth como tecnologia de comunicação, é possível simplificar a informação que será contida na *tag* associada ao dispositivo, como será explicado.

A informação trocada entre os dispositivos é baseada em mensagens NDEF [14]. Uma mensagem NDEF não é mais que um conjunto de NDEF Records. Um NDEF Record pode também encapsular outros NDEF Records. A estrutura é a definida na figura 5.9.

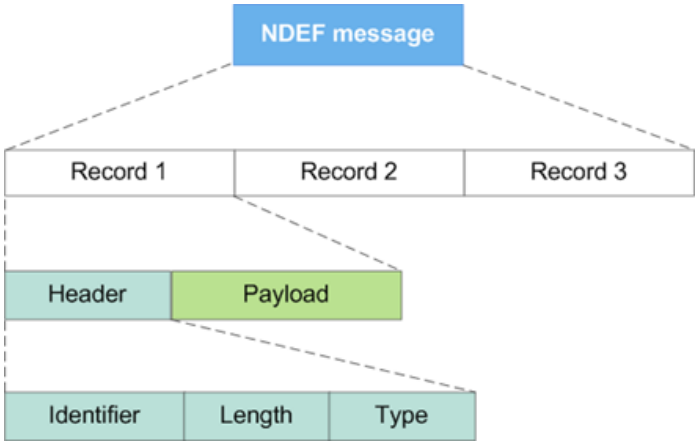


Figura 5.9: Estrutura de uma NDEF Message (imagem retirada de [13]).

A estrutura de um NDEF Record é a seguinte:

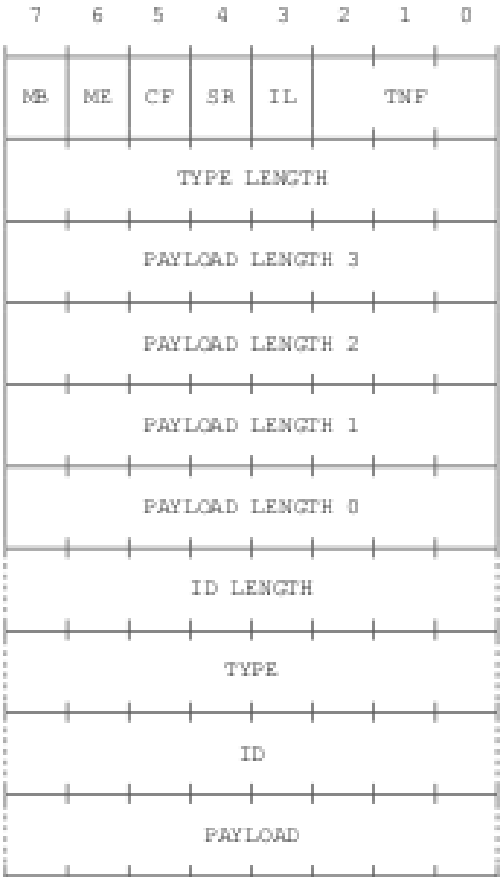


Figura 5.10: Estrutura de um NDEF Record (imagem retirada de [14]).

O significado de cada um dos campos é o seguinte:

- Header
  - MB (Message Begin)
    - Campo de 1 bit indicando que este é o primeiro Record da NDEF Message.
  - ME (Message End)
    - Campo de 1 bit indicando que este é o último Record da NDEF Message.

  - CF (Chunk Flag)
    - Campo de 1 bit usado para indicar Records fragmentados.

  - SR (Short Record)
    - Campo de 1 bit utilizado para indicar o tamanho do campo Payload Length. Se ativado, o campo Payload Length tem o tamanho de 1 byte, caso contrário, tem 4 bytes.

  - IL (ID Length Present)
    - Campo de 1 bit que indica se os campos ID e ID Length estão presentes neste Record. Se ativado, estão presentes, senão, estão omitidos.

  - TNF (Type Name Format)
    - Campo de 3 bits que indica a estrutura do campo Type.

- Type Length
  - Octeto que indica o tamanho, em bytes, do campo Type.

- Payload Length
  - Indica o tamanho, em bytes, do campo Payload. Se a *flag* SR estiver ativada, este campo contém um *unsigned integer* de 8 bits, senão contém um *unsigned integer* de 32 bits.

- ID Length
  - Octeto que indica o tamanho, em bytes, do campo ID.

- Type
  - Identifica o tipo de *payload* deste NDEF Record.

- ID
  - Identificador sob a forma de um URI.

- Payload
  - Transporta dados para serem interpretados a nível aplicacional. Como já foi referido, um NDEF Record poderá encapsular outros NDEF Records.

Esta é a estrutura geral de um NDEF Record. O *payload* de um Record poderá conter qualquer tipo de dados.

Bluetooth OOB Data Record

O NFC Forum definiu [15] um tipo de Record de modo a transportar a informação necessária para dar suporte ao mecanismo de emparelhamento OOB. Este Record, denominado Bluetooth OOB Data Record, é identificado no campo `Type` pela string `"application/vnd.bluetooth.ep.oob"`e define os dados que deve incluir no campo `Payload`.

Os dados são os que se apresentam na figura 5.11.

Name	Offset (Octets)	Size	Mandatory / Optional
OOB Data Length	0	2 octets	M
Bluetooth Device Address	2	6 octets	M
OOB Optional Data	8	N octets	O

Figura 5.11: Payload de um Bluetooth OOB Data Record (imagem retirada de [15]).

Ou seja, o *payload* de um NDEF Record terá, no mínimo, 8 bytes. O campo `OOB Data Length` ocupa 2 bytes e indica o tamanho, em bytes, ocupado por si mesmo

e pelos outros campos. O valor deste campo é igual ao valor do campo `Payload Length`. O campo `Bluetooth Device Address`, que ocupa 6 bytes, contém o endereço do dispositivo que se pretende emparelhar (por exemplo, no caso do *CareStore*, este campo será preenchido com o endereço Bluetooth do dispositivo médico a que se refere a *tag* NFC). Estes são os dois campos obrigatórios neste tipo de Record.

Relativamente ao último campo, `OOB Optional Data`, este é opcional, e deve ser incluído se for pretendido fornecer mais informação relativamente ao dispositivo que pretende ser emparelhado com o sistema. Essa informação pode ser, por exemplo, o seu nome, uma lista de serviços que disponibiliza ou a classe de dispositivo em que se insere. Naturalmente, o tamanho deste campo é variável. Cada um dos elementos presentes neste campo está inserido numa estrutura LTV, tal como apresentado na tabela 5.2.

Designação	Tamanho (bytes)	Descrição
Length	1	Tamanho ocupado, em bytes, pelos campos Type e Value.
Type	1	Identifica a informação contida em Value.
Value	Length - 1	Informação relativa ao dispositivo.

Tabela 5.2: Estrutura LTV.

O octeto inserido no campo `Type` irá conter um dos valores da lista apresentada na tabela 5.3.

Valor do campo `Type`	Descrição
0x02 - 0x07	Lista de serviços do dispositivo.
0x08 ou 0x09	Nome do dispositivo Bluetooth.
0x0D	Classe do dispositivo.
0x0E	Simple Pairing Hash C. Utilizado no Negotiated Handover.
0x0F	Simple Pairing Randomizer R. Utilizado no Negotiated Handover.

Tabela 5.3: Dados opcionais OOB.

Se houver necessidade de incluir vários destes campos opcionais, cada um será incluído numa estrutura LTV e agregado ao *payload* do Bluetooth OOB Data Record.

Implementação no *CareStore*

Naturalmente, no âmbito do projeto *CareStore*, foi implementada uma biblioteca de *software*para implementar os conceitos apresentados. Um dos focos desta biblioteca foi dar ao programador um modo simples de manipular os campos dos NDEF Records envolvidos num esquema de emparelhamento OOB. Para além disso, apesar de o projeto *CareStore* apenas implementar Static Handover, é importante garantir que o código é facilmente extensível para suportar Negotiated Handover. Esta biblioteca (escrita em `C++`) está organizada como mostrado na figura 5.12.

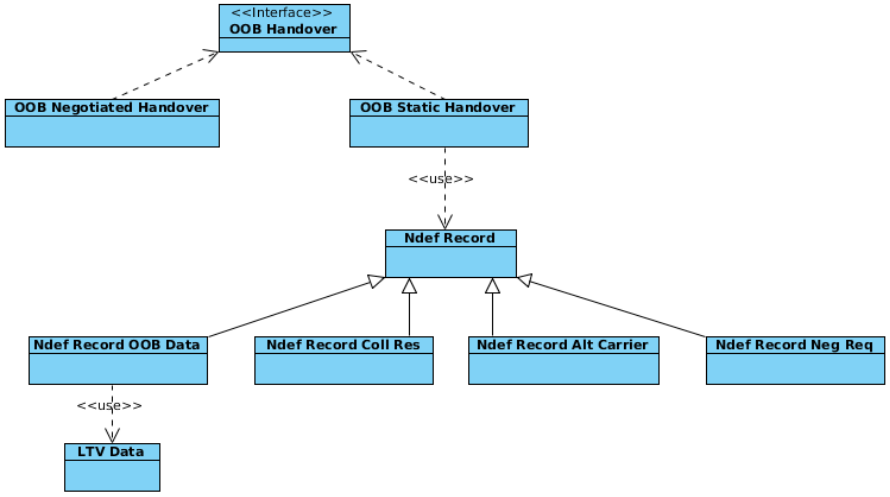


Figura 5.12: Estrutura da biblioteca de emparelhamento OOB.

A biblioteca criada é totalmente independente do *hardware* utilizado, sendo apenas uma ferramenta para a manipulação dos dados contidos num NDEF Record. No entanto, de maneira a tornar esta biblioteca mais poderosa, foram criados mecanismos para serializar os dados de um NDEF Record e realizar o *parsing* de um *array* de bytes para um NDEF Record. O objetivo é simplificar a interação com as API's de escrita e leitura de dispositivos NFC.

Portanto, existem dois métodos que cada classe de Record deve redefinir:

- Parse
  - Este método permite interpretar um *array* de bytes, de modo a criar um Record com os campos preenchidos. Isto é bastante útil pois coloca os dados num estado fácil de manipular.
- Serialize
  - Este método permite serializar os dados contidos num NDEF Record para um *array* de bytes. Deste modo, os dados poderão facilmente ser escritos para uma *tag* ou cartão NFC.

Estas operações facilitam a integração desta biblioteca num qualquer projeto, na medida em que criam um nível de abstração completo relativamente ao *hardware*.

Também as classes `OOB Negotiated Handover` e `OOB Static Handover` implementam estes métodos, sendo que, na sua implementação, invocam os métodos de `Parse` e `Serialize` dos tipos de NDEF Record que implementam. No caso de uma operação de Static Handover no projeto *CareStore*, os dispositivos médicos apenas possuem mecanismos de comunicação Bluetooth, de modo que é possível simplificar os Records envolvidos, sendo apenas necessário escrever um `Bluetooth OOB Data Record` (implementado na classe `Ndef Record OOB Data`), com os dados do dispositivo em questão. Ou seja, para serializar toda a informação de uma operação de Static Handover apenas é necessário usar o método `Serialize` de uma classe, no entanto, uma implementação de Negotiated Handover, iria invocar as operações de múltiplos tipos de Record.

Para terminar, é interessante fazer uma análise da quantidade de informação necessária para pôr em prática este mecanismo. Isto é especialmente importante para a seleção de dispositivo NFC, pois alguns tipos de *tag* NFC tem restrições limitadoras, no que toca a memória disponível. É possível analisar os dados que podem ser encontrados num esquema de Static Handover, onde os dispositivos apenas tem capacidades de comunicação via Bluetooth:

Campo	Tamanho (em bytes)	Mandatory/ Optional	Observações
NDEF Record Header	1	M	—
Type Length	1	M	O valor é sempre 32.
Payload Length	1	M	—
Type	32	M	O valor é sempre "*application/vnd.bluetooth.ep.oob*".
OOB Data Length	2	M	—
Bluetooth Device Address	6	M	—
Bluetooth Local Name	$2 + N$	O	N = Tamanho do Nome, $N <= 20$
Class of Device	$2 + 3$	O	—
Service List	$2 + (M$ x $2)$	O	M = Número de Serviços

Tabela 5.4: Dados do Static Handover no *CareStore*.

Estes dados compõem a informação que será escrita nas *tags* NFC do sistema *CareStore*, que não sendo muito, é facilmente suportada por praticamente todos os tipos de *tag* ou cartões NFC. Fazendo uma análise rápida aos valores exatos que podemos esperar:

- Sem dados opcionais:
  - 43 bytes;
- Com dados opcionais, exceto lista de serviços:
  - $50 + N$ bytes;
  - O valor máximo é 70 bytes, assumindo que o Nome do dispositivo pode ter 20 caracteres, no máximo;
- Com dados opcionais, incluindo lista de serviços:
  - $52 + N + (M * 2)$ bytes;
  - $72 + (M * 2)$ bytes, se o Nome do dispositivo tiver o número de caracteres máximo;

– Assumindo que o dispositivo publicita 10 serviços, ainda assim apenas teriam de ser escritos 92 bytes na *tag* NFC, o que é facilmente suportado na maioria das *tags* no mercado e mais que suficiente para o projeto *CareStore*.

Para terminar esta parte, é necessário salientar que as *tags* usadas no desenvolvimento do projeto, as NTAG203, disponibilizam 144 bytes de memória, o que é mais que suficiente para o esquema proposto.

5.1.4 Comunicação com dispositivos médicos

O objetivo principal do *CareStore* sempre foi a recolha dos dados de dispositivos médicos. Para esse efeito, a tecnologia Bluetooth disponibiliza um perfil (HDP) que facilita a troca de dados médicos entre dois dispositivos que o implementem. No caso do projeto *CareStore*, um destes dois dispositivos é sempre o CRIP, enquanto o outro é um dispositivo médico com capacidades Bluetooth, certificado pela Continua Health Alliance[1]. Esta certificação valida o modelo de dados utilizado no dispositivo, de modo a garantir a interoperabilidade entre dispositivos de fabricantes diferentes. Os membros desta entidade constituem os principais fabricantes de *hardware* no mercado (IBM, Intel, Samsung, Cisco Systems, etc). Os dispositivos utilizados no desenvolvimento deste projeto são certificados por esta entidade.

É importante fazer uma pequena introdução às tecnologias envolvidas nesta funcionalidade do projeto *CareStore*, na medida em que já ultrapassa as funcionalidades básicas Bluetooth apresentadas até agora.

5.1.4.1 HDP

O HDP trata-se de um perfil aplicacional utilizado para criar canais de comunicação Bluetooth entre dispositivos médicos e dispositivos de recolha de dados. A pilha Bluetooth BlueZ implementa este perfil, garantindo aos programadores um modo fácil de criar conexões HDP entre dispositivos que implementem o perfil. Esta camada insere-se na pilha Bluetooth tal é como apresentado na figura 5.13.

[1] `http://www.continuaalliance.org/`

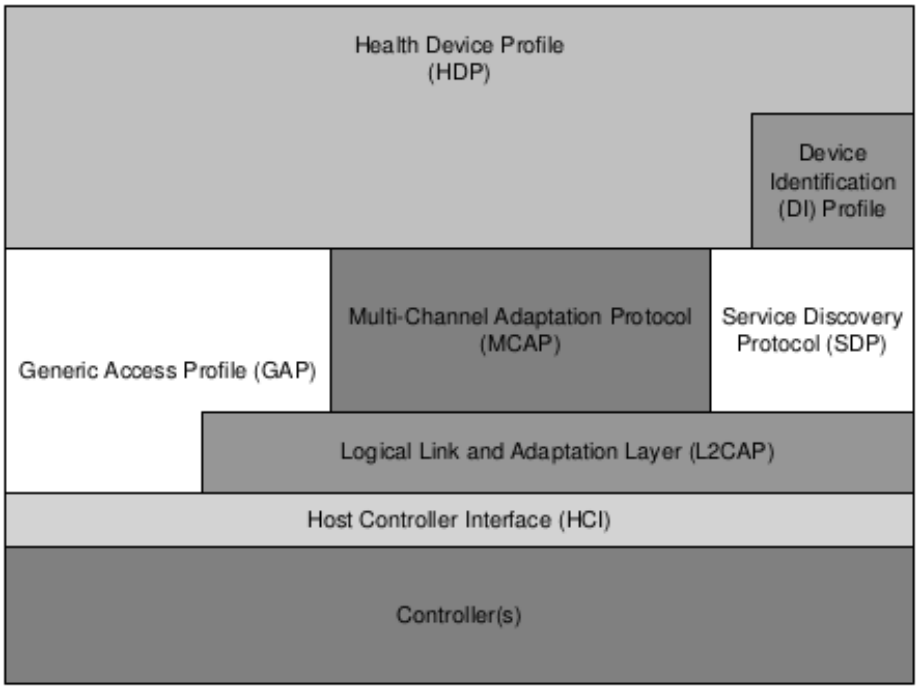


Figura 5.13: HDP na *stack* Bluetooth (imagem retirada de [16]).

No HDP não existem clientes nem servidores. Um dispositivo médico que trata de recolher os dados e os disponibilizar, denomina-se *Source*, enquanto que um dispositivo responsável por receber e armazenar esses dados é chamado *Sink*. Geralmente, cabe a uma *Source* iniciar uma conexão a um *Sink*, visto que são os dispositivos que têm a capacidade de detetar quando uma amostra é recolhida. Naturalmente, isto pressupõe que um *Sink* é um dispositivo que não possui limitações em termos de energia. Este processo é melhor explicado neste exemplo retirado da especificação do HDP.

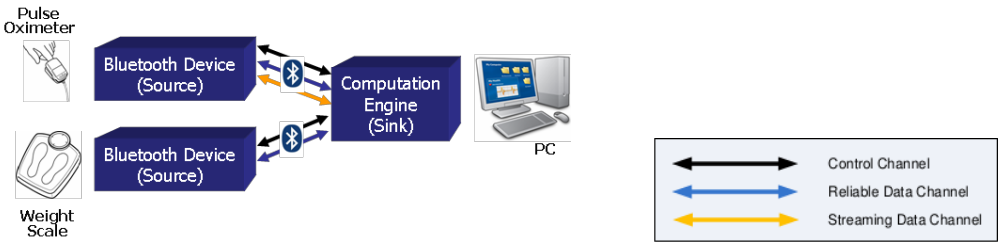
> *"A blood pressure unit (Source) is used to measure blood pressure. After measuring the result, the Source searches for and discovers the Sink device that is to receive its blood pressure data. The Source initiates a connection, and the application data is transferred. In this case, the Source is the Initiator, and the Sink is the Acceptor."* [16]

As conexões entre dispositivos são asseguradas pela camada MCAP, que permite

o estabelecimento de canais de comunicação L2CAP. É importante referir que, na realidade, numa conexão HDP entre dois dispositivos, não existe apenas um canal de comunicação. Quando é estabelecida uma primeira conexão é criado aquilo a que se chama um *Control Channel*, que por sua vez irá permitir a criação de *Data Channels*, por onde são passados efetivamente os dados. É obrigatório existir um e um só *Control Channel* e terá de existir pelo menos um *Data Channel*. Os *Data Channels* podem ser de dois tipos:

- *Reliable*
  - Num processo de comunicação HDP tem de existir pelo menos um destes canais. Neste tipo de canal são enviados, esporádica ou periodicamente, dados aplicacionais;
- *Streaming*
  - Neste tipo de canal são enviados, continuamente, dados aplicacionais.

Para terminar, na imagem 5.14 é apresentado um caso exemplo, retirado da especificação HDP.



(a) (b)

Figura 5.14: Exemplo de conexões HDP (imagens retiradas de [16]).

5.1.4.2 IEEE 11073

A sigla IEEE 11073 representa um conjunto de normas que definem o modelo de dados utilizado pelos dispositivos médicos, para o armazenamento e troca de dados

médicos. Estas normas são essenciais, na medida em que garantem a interoperabilidade entre dispositivos de diferentes fabricantes. A Continua Health Alliance aconselha a utilização destas especificações aos seus membros. É importante salientar que estas especificações nada têm a ver com tecnologias Bluetooth, visto que esta é apenas a tecnologia de comunicação usada, o que deve ser totalmente independente de como os dados são armazenados ou qual o modelo de dados utilizado para que estes sejam comunicados eficientemente, que é o que este conjunto de normas define.

Alguns exemplos de documentos que fazem parte do conjunto de normas IEEE 11073:

- IEEE 11073-20601 - Optimized Exchange Protocol - Define o protocolo utilizado para troca de dados entre dispositivos;
- IEEE 11073-10404 - Específica o modelo de dados utilizado num monitor de pressão sanguínea;
- IEEE 11073-10415 - Específica o modelo de dados utilizado numa balança;
- IEEE 11073-10406 - Específica o modelo de dados utilizado por um dispositivo de eletrocardiografia.

Para além destas, existem muitas mais normas para o mais variado tipo de dispositivos médicos. Tal como no HDP, também aqui existe uma nomenclatura específica para as entidades do processo de comunicação. No âmbito do IEEE 11073, existe o conceito de *Agents* e *Managers*. Um *Agent* trata-se de um dispositivo médico e é geralmente um dispositivo com limitações em termos de bateria ou mesmo interface com o utilizador. Relativamente ao *Manager*, este é um dispositivo com mais poder computacional e é tido como não tendo limitações de energia. Transpondo para os termos usados relativamente ao HDP, é possível dizer que um *Agent* é a mesma coisa que uma *Source* e que um *Manager* tem características iguais aos de um *Sink*.

Em termos de localização na *stack* Bluetooth, é possível ver uma implementação destes *standards* como que formando uma camada de *software* acima do HDP.

5.1.4.3 Implementação no CareStore

Tendo em conta o elevado preço das normas IEEE 11073, no *CareStore* foi utilizada uma ferramenta que implementa alguns destes *standards* e fornece uma camada de ligação com a implementação HDP da BlueZ, de modo a implementar os canais de troca de dados e o canal de controlo. Esta ferramenta, denominada *Antidote*, garante abstração, no sentido em que não obriga o programador a conhecer o modelo de dados dos dispositivos que implementa e na medida em que abstrai a API DBus fornecida pela BlueZ para a programação de funcionalidades HDP. Os dispositivos suportados pelo Antidote são:

- IEEE 11073-10404 - Oximetro de pulsação cardíaca;
- IEEE 11073-10407 - Monitor de pressão sanguínea;
- IEEE 11073-10415 - Balança;
- IEEE 11073-10417 - Sensor de glicose.

O Antidote fornece duas API para serem utilizadas pelos programadores.

Uma API na linguagem C, que permite ao programador estabelecer qual a tecnologia de transporte da informação (no caso do *CareStore*, seria Bluetooth), estabelecer qual o tipo de dispositivos com que é prendido estabelecer comunicação ou mesmo emular o comportamento e modelo de dados de um dispositivo médico, visto que permite a implementação de *Agents* e *Managers* IEEE 11073.

Para além desta API, com o Antidote é fornecido um *daemon*, chamado `healthd`, que exporta uma API DBus, à semelhança do que é possível encontrar na BlueZ. Esta constitui a segunda API e é a que é efetivamente utilizada no desenvolvimento do projeto *CareStore*.

Este *daemon* implementa um *Manager*, que tem como objetivo recolher dados de *Agents*, e permite o acesso aos dados enviados por estes através do uso de uma API DBus. Isto é bastante atrativo, na medida em que permite que seja possível criar uma aplicação numa qualquer linguagem, desde que essa linguagem garanta mecanismos que suportem o DBus.

No projeto *CareStore*, de modo a implementar a recolha de amostras dos dispositivos médicos, foi exatamente isto que foi feito. O *software*do CRIP interage com o

`healthd` via DBus, de modo a ser possível tratar os dados provenientes dos dispositivos médicos no próprio código do CRIP.

De modo a habilitar o `healthd` a receber dados de dispositivos médicos, é necessário utilizar o seguinte método, pertencente à interface Manager do objeto DBus implementado pelo `healthd`:

- `void ConfigurePassive(object agent, int data_types[])`

A invocação deste método faz com que o *daemon* esteja à escuta de conexões de *Agents*. O tipo de dispositivos que se podem conectar pode ser definido no segundo parâmetro do método. No caso do *CareStore*, este método é invocado de modo a habilitar todos os tipos de dispositivos suportados pelo Antidote. Relativamente ao primeiro parâmetro do método, deve ser especificado o Object Path do objeto DBus que identifica o objeto para onde serão enviados os dados dos *Agents* que estabelecem comunicação com o `healthd`. Para o *CareStore*, foi definido o Object Path: `/com/signove/health/agent/carestore`.

A estratégia a seguir é bastante semelhante à utilizada no caso do emparelhamento, ou seja, foi criado um objeto DBus, implementando um conjunto de métodos pertencentes a uma interface Agent DBus que são invocados pelo *daemon*. Essa interface possui bastantes métodos, mas apenas a implementação de alguns é relevante para o projeto *CareStore*:

- `void Connected(object device, string address)`
  - Este método é invocado quando um *Agent* se conecta, através do HDP, ao CRIP. Um dos parâmetros enviados ao método é o endereço do dispositivo;
- `void Associated(object device, string jsondata)`
  - Este método é invocado quando um *Agent* atinge um estado de associação com o CRIP, estando então preparado para enviar dados médicos. O segundo parâmetro do método transporta uma string JSON, contendo os dados de associação do dispositivo;
- `void MeasurementData(object device, string jsondata)`

  - Este método é invocado quando os dados médicos são enviados para o CRIP. O segundo parâmetro do método transporta uma string JSON, contendo os dados médicos, correspondentes a amostras recolhidas pelo *Agent*;

- `void Disassociated(object device)`
  - Este método é invocado quando um *Agent* se dissocia do CRIP, deixando de poder comunicar dados médicos;

- `void Disconnected(object device)`
  - Este método é invocado quando um *Agent* se desconecta do CRIP.

Todos estes métodos transportam um parâmetro em comum, que é o Object Path de um objeto DBus referente ao dispositivo ao qual o método invocado se refere. Isto acontece pois o *Manager*, neste caso o *daemon* `healthd`, implementa uma interface DBus que permite realizar operações sobre os *Agents* que se conectam a ele. No caso do *CareStore*, o uso dessa interface não foi relevante, no entanto, estes Object Path foram utilizados para poder estabelecer a relação entre a invocação dos diferentes métodos, associando-os ao mesmo dispositivo. Esta prática é a recomendada pelos criadores do Antidote.

É importante perceber todos os estados pelos quais um *Agent* pode passar, no tempo de vida de um processo de comunicação com o Antidote:

- Disconnected
- Connected
  - Associating
  - Associated
  - Disassociating
  - Unassociated

Partindo de um estado *Unassociated* é possível um *Agent* voltar a associar-se ou então desconectar-se. Os métodos DBus da interface Agent implementados, ajudam a perceber qual é o estado em que se encontra o *Agent*. De modo a identificar qual o dispositivo a que se refere a invocação de um método, tal como já foi explicado, é utilizado o Object Path enviado com os métodos. No CRIP é mantido o estado de um *Agent* bem como um registo das amostras enviadas por um dispositivo ao longo do seu processo de comunicação. Também é armazenado o tempo em que cada amostra foi recolhida, visto que, embora as amostras transportem indicação temporal de quando a mesma foi efetuada, esta é baseada num relógio interno do dispositivo e os relógios dos dispositivos não estão sincronizados.

Este nível de gestão permite resolver alguns problemas:

- Permite garantir que uma amostra possui dados atualizados, visto que bastantes dispositivos médicos implementam um sistema de armazenamento de amostras no caso de não existir um *Manager* que recolha as amostras em tempo real, fazendo com que estas sejam enviadas todas ao mesmo tempo, assim que o *Agent* se associa ao *Manager*;

- Evita que sejam aceites dados médicos de um dispositivo que não está num estado onde esses dados possam ser considerados válidos. Por exemplo, não faz sentido receber dados de um dispositivo que não se conectou ou associou.

Para além disto, existe uma grande vantagem nesta abordagem. Como foi desenvolvido um módulo independente para fazer a gestão da comunicação entre o CRIP e os dispositivos médicos, se for garantido o acesso aos dados recolhidos à API do CRIP, é possível o desenvolvimento de métodos que respeitem as necessidades de cada aplicação. Por exemplo, no CRIP, de modo a ler uma amostra de um dispositivo médico tem de ser feita uma chamada explícita ao método que implementa essa funcionalidade (como será explicado na secção 5.1.5), que irá escutar o módulo que faz a gestão da comunicação entre *Manager* e *Agents*. Esse método implementa um *timeout*, de modo que se o dispositivo do qual se pretende recolher uma amostra não enviar nenhuma amostra durante um certo período de tempo, o método retorna sem nenhuma amostra recolhida para o dispositivo especificado. No entanto, durante esse

intervalo de tempo, até podem ser recebidas amostras de outros dispositivos diferentes, sendo que essas amostras são registadas. Simplesmente, como as amostras não se referem ao dispositivo pretendido, são ignoradas pela API do CRIP.

O algoritmo desenvolvido de modo a implementar a leitura de dispositivos médicos é descrito na figura 5.15.

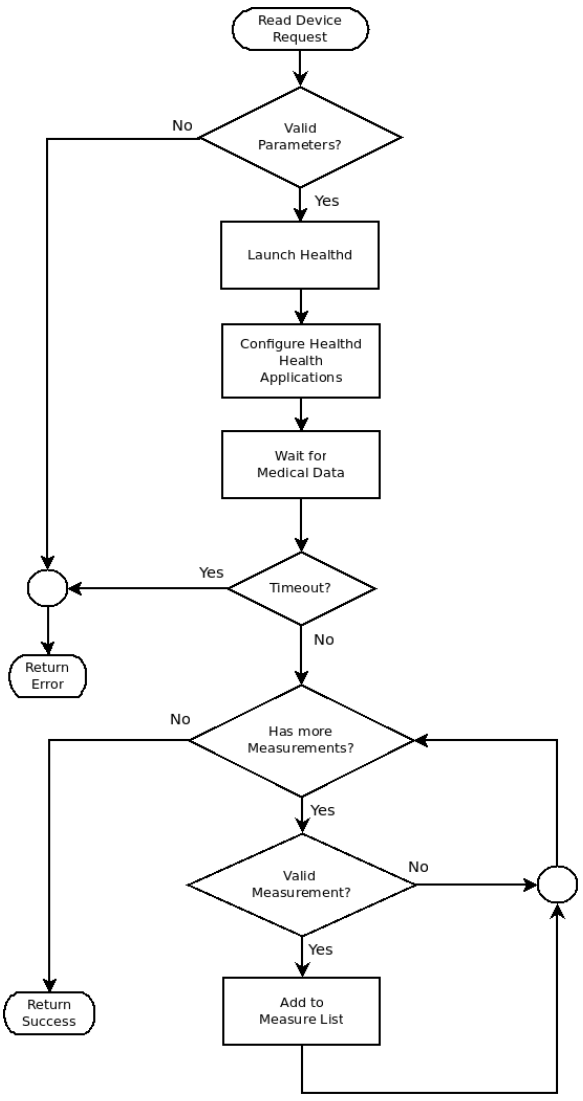


Figura 5.15: Algoritmo de leitura de dados médicos de dispositivos Bluetooth no 3º protótipo.

Para terminar, é importante também fazer a relação da máquina de estados do Antidote com o método `ReadDevice`. Isso é explicado no diagrama 5.16.

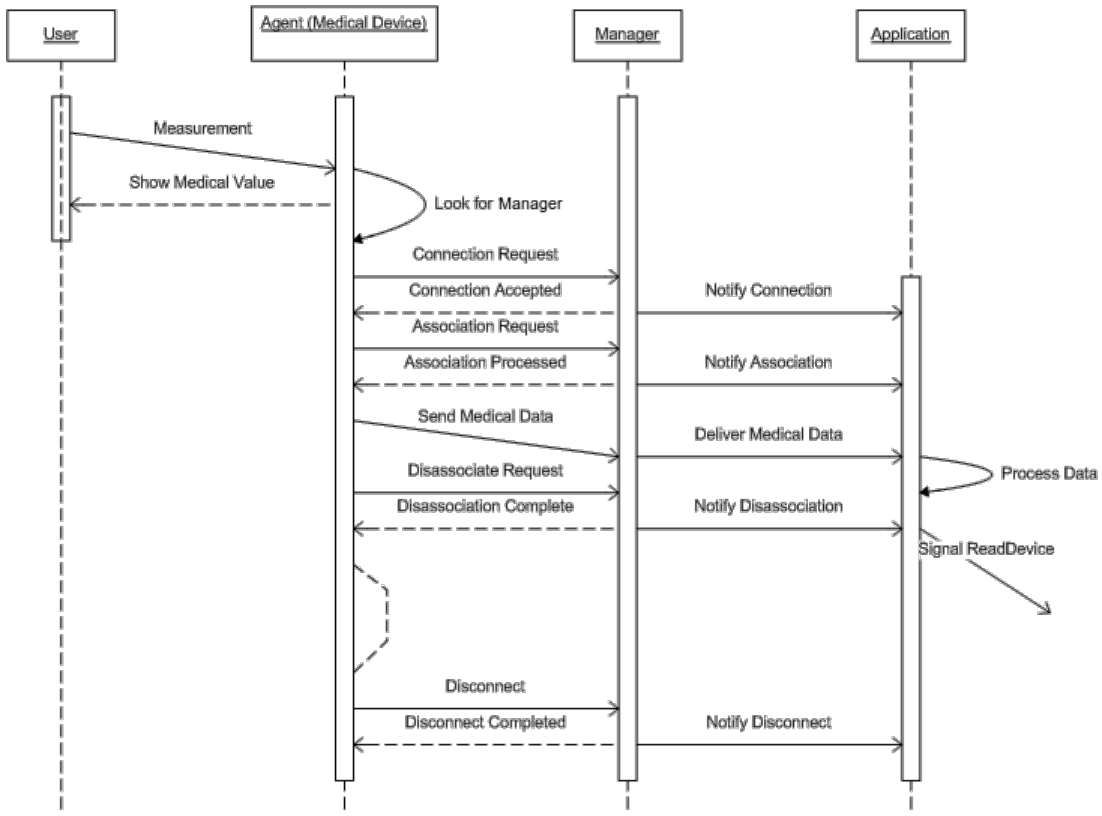


Figura 5.16: Exemplo da leitura de um dispositivo com o Antidote.

A figura 5.16 é bastante simples de seguir, no entanto é importante fazer a correspondência entre os estados da máquina de estados implementada pelo Antidote e a aplicação desenvolvida para o *CareStore*.

Quando um utilizador utiliza um dispositivo médico (*Agent*) para tirar uma amostra de um dado parâmetro de saúde, o valor medido geralmente é revelado por via de um visor ao utilizador. No caso deste tipo de dispositivos com capacidades Bluetooth, quando é tirada uma amostra, este tenta procurar um *Manager* (no projeto *CareStore*, o *Manager* será o CRIP), tal como definido nas normas IEEE 11073. Se este não for descoberto, a amostra é armazenada ou descartada, consoante a preferência do fabricante. Se o *Manager* for encontrado, é estabelecida uma conexão e o dispositivo é associado ao CRIP. Estas duas mudanças de estado, conexão e associação, são transmitidas ao módulo Bluetooth do projeto *CareStore*.

Só depois de *Agent* e *Manager* estarem associados é que é feita a transmissão dos dados propriamente ditos, podendo em seguida ocorrer a dissociação e desconexão

entre os dois elementos. Naturalmente que essas mudanças de estado são igualmente comunicadas ao módulo Bluetooth.

Algo importante que deve ser referido é que os dados da amostra não são tratados no CRIP, sendo enviada para o CAALHP a *string* JSON tal e qual como é recebida do `healthd`.

5.1.5 CRIP Bluetooth API

Tendo sido explicada a implementação de todas as funcionalidades Bluetooth do projeto *CareStore*, é importante explicitar quais os métodos que compõem a API Bluetooth do CRIP. Esta API foi desenvolvida em C++ e disponibiliza os seguintes métodos:

- ICripBluetooth::EnumState ScanDevices(std::vector<CripBluetoothDevice*>* devices)
- ICripBluetooth::EnumState GetDevices(std::vector<CripBluetoothDevice*>* devices)
- ICripBluetooth::EnumState PairDevice(const std::string& btAddress)
- ICripBluetooth::EnumState PairDevice(OobHandover* record)
- ICripBluetooth::EnumState ReadDevice(const std::string& btAddress, std::vector<std::string>* measure)

Como é possível verificar, a API desenvolvida é bastante simples, no entanto cobre todas as necessidades atuais do projeto *CareStore*.

Antes de explicar cada um dos métodos, é importante perceber o tipo de valor que devolvem. De modo a facilitar a utilização desta API, foi definido um conjunto de valores de retorno que facilitam a compreensão de qual o resultado da execução de um método. Os valores de retorno possíveis são indicados na tabela 5.5.

Código Retorno	Valor	Descrição
BT_OK	0	Valor que indica o sucesso da operação.
BT_ERROR_OTHER	-1	Valor devolvido para erros genéricos.
BT_ERROR_ADAPTER _NOT_FOUND	-2	Valor devolvido quando o BT111 não é detetado.
BT_ERROR_NO _PAIRED_DEVICES	-3	Valor devolvido quando não existem dispositivos emparelhados com o CRIP.
BT_ERROR_TIMEOUT	-4	Valor devolvido quando existe *timeout* na leitura de dados médicos de um dispositivo.
BT_ERROR_HCI _CONNECTION	-5	Valor devolvido quando não é possível estabelecer uma conexão HCI com o BT111.
BT_ERROR_NO _SCANNED_DEVICES	-6	Valor devolvido quando uma descoberta de dispositivos Bluetooth não deteta nenhum dispositivo.
BT_ERROR_DBUS _NO_CONNECTION	-7	Valor devolvido quando não é possível estabelecer conexão com o *daemon* DBus.
BT_ERROR_DBUS _PROXY_CREATION	-8	Valor devolvido quando não é possível criar um *proxy* para um objeto DBus.
BT_ERROR_DBUS _PROXY_METHOD	-9	Valor devolvido quando não é possível invocar um método de um objeto DBus.
BT_ERROR_DBUS _BAD_MESSAGE	-10	Valor devolvido quando é detetado um erro numa mensagem DBus.
BT_ERROR_DBUS _GENERIC	-11	Valor devolvido para erros genéricos relacionados com DBus.

Código Retorno	Valor	Descrição
`BT_ERROR_DBUS _GENERIC`	-11	Valor devolvido para erros genéricos relacionados com DBus.
`BT_ERROR_DBUS _OPATH_IN_USE`	-12	Valor devolvido para quando se tenta registar um Object Path que já está a ser usado.
`BT_ERROR_BAD _PARAMETERS`	-13	Valor devolvido quando os parâmetros de um método não são válidos.
`BT_ERROR_ALREADY _PAIRED`	-14	Valor devolvido para indicar que se está a tentar emparelhar um dispositivo já emparelhado.
`BT_ERROR_NOT _PAIRED`	-15	Valor devolvido para indicar o insucesso de uma operação de emparelhamento.
`BT_ERROR_HEALTHD`	-16	Valor devolvido indicando insucesso a iniciar o *daemon* `healthd`.
`BT_ERROR_PIPE _CREATION`	-17	Valor devolvido para indicar um erro na criação de um *pipe* para sinalização de emparelhamento de dispositivos Bluetooth.

Tabela 5.5: Valores devolvidos pelos métodos da API Bluetooth do CRIP.

Relativamente aos métodos em si, é necessário analisar os parâmetros que incluem.

O método `ScanDevices` permite a descoberta de dispositivos Bluetooth nas imediações do CRIP e recebe como parâmetro um endereço a apontar para uma lista que é preenchida com os dispositivos encontrados num processo de descoberta de dispositivos.

O método `GetDevices` segue a mesma lógica do método anterior, sendo passado como parâmetro um endereço a apontar para uma lista que é preenchida com os dispositivos emparelhados com o CRIP e que se encontram em atividade nas imediações.

Relativamente ao emparelhamento de dispositivos, são fornecidos dois métodos. O primeiro método é sempre invocado, sendo que o segundo método recebe como parâmetro um objeto que contém os NDEF Records envolvidos num processo de *Handover*. Daí é retirado o endereço do dispositivo Bluetooth com o qual é pretendido que o CRIP se emparelhe e é invocado o primeiro método, que recebe como parâmetro uma *string* com o endereço do dispositivo Bluetooth.

O método `ReadDevice` procede à leitura de uma amostra de um dispositivo médico. Como primeiro parâmetro é passada uma *string* com o endereço do dispositivo Bluetooth do qual é pretendido receber uma amostra de dados médicos. Relativamente ao segundo parâmetro, é passada a lista de amostras que deve ser preenchida com as amostras do dispositivo.

Na leitura de dados médicos de um dispositivo, é necessário prever a leitura de múltiplas *strings* JSON, visto que alguns dispositivos servem para medir vários parâmetros de saúde. Por exemplo, o monitor de pressão sanguínea utilizado no desenvolvimento do projeto *CareStore*, quando realiza uma medição, envia uma amostra com o valor da pulsação e outra com os índices de pressão sanguínea. No entanto, o conjunto desses dados compõe apenas uma amostra.

5.1.6 SO da plataforma CRIP

Na entrega do segundo e terceiro protótipo do CRIP, foi necessário preparar uma imagem de um sistema operativo com todas as dependências de *software* necessárias à execução do código da plataforma. Para esse efeito foi utilizada uma implementação de um sistema operativo baseado no Raspbian para a Raspberry Pi[2]. Esta versão apresenta melhorias de desempenho relativamente ao Raspbian e é distribuída apenas com os pacotes essenciais ao funcionamento do sistema operativo.

Para dar suporte ao *software* e *hardware* do CRIP, foram instaladas as seguintes ferramentas:

- BlueZ;
- DBus;
- libNFC;
- Glib;
- Antidote;
- libJSON;
- monit;
- microhttpd;
- sqlite3;
- isc-dhcp-server.

Algumas destas ferramentas já foram mencionadas ao longo deste documento, de modo que é fácil perceber onde se inserem no projeto. Outras são referentes a componentes do projeto ou simplesmente utilidades para uma fácil utilização do CRIP.

Por exemplo, o `monit` é uma ferramenta bastante interessante, na medida em que permite monitorizar recursos ou aplicações e atuar perante algumas condições configuráveis pelo utilizador. Neste caso, é pretendido garantir que o *daemon* que agrega

[2] `http://www.linuxsystems.it/raspbian-wheezy-armhf-raspberry-pi-minimal-image/`

o *software* do CRIP é monitorizado, de modo a ser reiniciado em caso de *crash*. Desse modo, o `monit` foi configurado para esse efeito.

O `sqlite` permite manter uma base de dados dos utilizadores registados com o *CareStore*, dando acesso às funcionalidades do sistema a utilizadores autorizados e que sejam devidamente identificados pelo CRIP.

Relativamente ao `microhttpd`, esta ferramenta é utilizada para dar a capacidade ao CRIP de responder a pedidos HTTP efetuados pelo CAALHP.

De modo a garantir uma filosofia *plug & play*, foi instalado um servidor DHCP que atribui um endereço IP à interface Ethernet do dispositivo a que o CRIP é ligado, de modo a não ter de ser configurado manualmente por quem instala o CRIP.

Naturalmente que há bastantes mais dependências que foi necessário instalar, no entanto, estas são as de maior relevo para o funcionamento do CRIP.

Tendo apresentado a implementação do CRIP, é possível apresentar os resultados obtidos com essa implementação e perceber o que é afinal o produto CRIP.

Capítulo 6

Resultados

Nesta secção são apresentados os resultados da implementação descrita no capítulo 5. Serão primeiro apresentados os protótipos criados para as diferentes fases do projeto *CareStore*. Em seguida são apresentados exemplos de alguns testes realizados sobre algumas das funcionalidades.

6.1 Protótipos do CareStore

Nesta secção são apresentados os protótipos desenvolvidos no âmbito do projeto *CareStore*.

6.1.1 1º e 2º Protótipo

Os dois primeiros protótipos são exatamente iguais em termos de características de *hardware*, sendo que as diferenças resultam do *software* que implementa, tal como já foi explicado no capítulo anterior. A imagem 6.1 demonstra o aspeto físico da plataforma, sendo que as suas características já foram explicadas no capítulo 3.

(a) Visão traseira do 2º protótipo do CRIP.

(b) Visão frontal do 2º protótipo do CRIP.

(c) Visão de cima do 2º protótipo do CRIP.

Figura 6.1: Imagens do 2º protótipo do CRIP.

6.1.2 3º Protótipo

O terceiro e atual protótipo contém também o mesmo *hardware* que os protótipos anteriores, sendo que a única diferença assenta na caixa envolvente, que identifica mais facilmente o módulo como pertencendo ao projeto *CareStore*. Mais uma vez, a diferença principal assenta no *software* com que é distribuído, tal como explicado no capítulo 3. A imagem 6.1 demonstra o aspeto físico da plataforma.

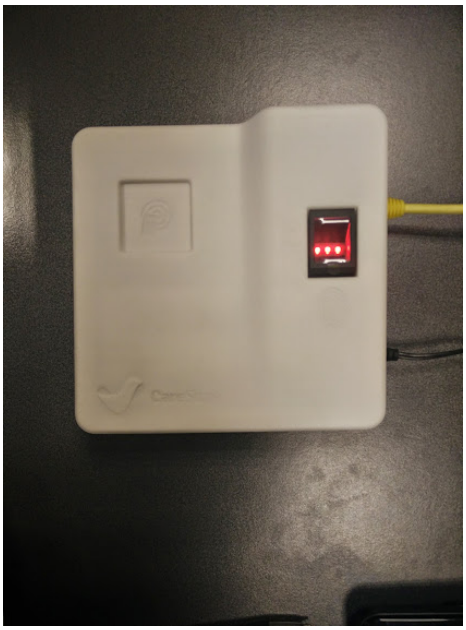


Figura 6.2: Visão de cima do 3º protótipo do CRIP.

Como é possível ver, o *design* desta última solução é bastante mais apelativo. Para terminar, a imagem 6.3 mostra o interior do CRIP, onde é possível identificar os componentes apresentados na secção 3.2.1.

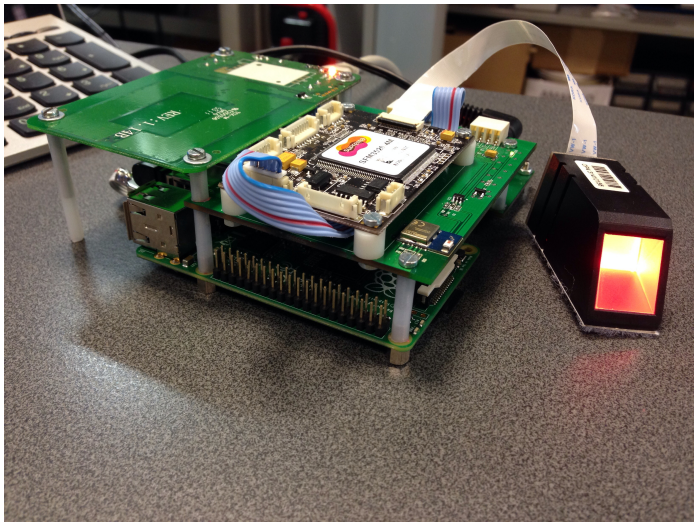


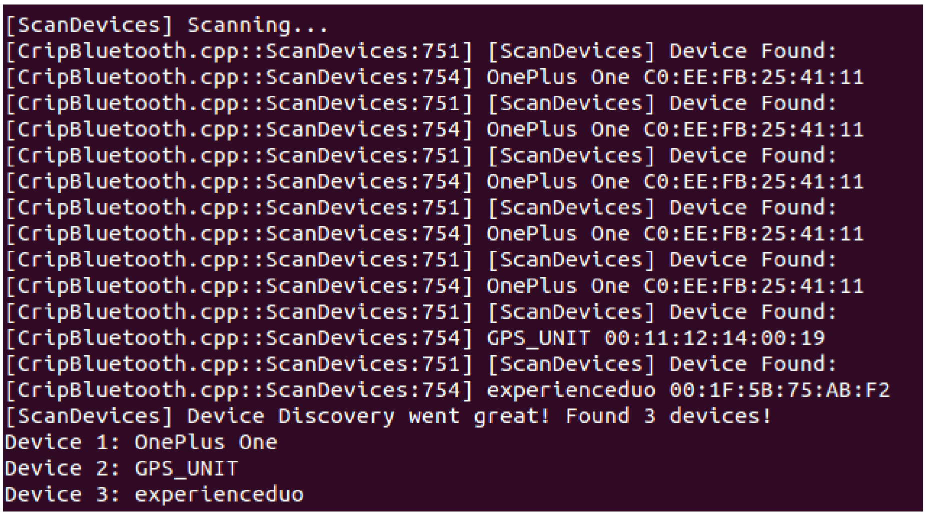
Figura 6.3: Interior do CRIP.

6.2 Testes ao Protótipo

Nesta secção são apresentados alguns testes às funcionalidades Bluetooth do CRIP, recorrendo a algumas aplicações exemplo criadas para realização de testes à plataforma. Como a integração com o CAALHP ainda não está finalizada, não é possível mostrar o *output* destas operações na sua versão final.

6.2.1 Descoberta de dispositivos

O primeiro teste efetuado foi o da invocação do método `ScanDevices`. A figura 6.4 apresenta o *output* dessa operação.



```
[ScanDevices] Scanning...
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] OnePlus One C0:EE:FB:25:41:11
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] OnePlus One C0:EE:FB:25:41:11
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] OnePlus One C0:EE:FB:25:41:11
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] OnePlus One C0:EE:FB:25:41:11
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] OnePlus One C0:EE:FB:25:41:11
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] GPS_UNIT 00:11:12:14:00:19
[CripBluetooth.cpp::ScanDevices:751] [ScanDevices] Device Found:
[CripBluetooth.cpp::ScanDevices:754] experienceduo 00:1F:5B:75:AB:F2
[ScanDevices] Device Discovery went great! Found 3 devices!
Device 1: OnePlus One
Device 2: GPS_UNIT
Device 3: experienceduo
```

Figura 6.4: Teste ao método `ScanDevices`.

Neste caso, foram encontrados três dispositivos Bluetooth nas imediações do CRIP, o que valida o bom funcionamento da operação. De notar que os dispositivos se podem anunciar mais que uma vez durante um processo de descoberta. É isso que é visível na figura 6.4.

6.2.2 Emparelhamento de dispositivos

Tendo em conta o *output* da operação anterior, foi tentado realizar uma operação de emparelhamento com o dispositivo chamado `OnePlus One`, utilizando o método `PairDevice`. De salientar que esta operação não utiliza o esquema de emparelhamento OOB via NFC. A figura 6.5 representa o *output* da operação de emparelhamento.

```
[PairDevice] Pairing with C0:EE:FB:25:41:11...
[PairDevice] Paired with C0:EE:FB:25:41:11!
```

Figura 6.5: Teste ao método `PairDevice`, sem uso de tecnologias NFC.

Tendo isto, é agora tentado o emparelhamento utilizando o esquema OOB via NFC. Na figura 6.6 podemos observar a *tag* NFC colocada numa balança `UC-321PBT-C`. O método utilizado é na mesma o `PairDevice`, no entanto, desta vez é passada como parâmetro do método a informação lida da *tag* NFC do dispositivo. Nessa *tag* foi previamente escrita a informação necessária ao emparelhamento do dispositivo (Endereço do dispositivo).

Figura 6.6: NTAG203 colocada na balança `UC-321PBT-C`.

Executando uma aplicação para realizar o emparelhamento utilizando o mecanismo OOB via NFC, temos o seguinte *output*, tal como demonstrado na figura 6.7.

```
[PairingTool] NFC Reader is loaded...
error   libnfc.driver.acr122_usb        PN532 didn't reply
error   libnfc.driver.acr122_usb        PN532 didn't reply
error   libnfc.driver.acr122_usb        PN532 didn't reply
error   libnfc.driver.acr122_usb        PN532 didn't reply
error   libnfc.driver.acr122_usb        PN532 didn't reply
[PairingTool] Detected NTAG203 Tag with ID: 04 E7 4E 3A BC 2B 80
[PairingTool] Device Address: 00:09:1F:80:58:D1
[PairingTool] Device Name: Jakob
[PairingTool] Device 00:09:1F:80:58:D1 was Paired!
```

Figura 6.7: Teste ao método `PairDevice`, utilizando emparelhamento OOB via NFC.

Os erros que se veem na figura 6.7 não se tratam de erros reais. Apenas simbolizam o facto de não ter sido detetada nenhuma *tag* NFC durante o tempo em que o programa esteve em execução antes de ser apresentada a *tag*.

Deste modo, é emparelhada a balança com o nome *Jakob*. A partir deste momento, o CRIP pode receber dados médicos deste dispositivo, como será possível observar.

6.2.3 Deteção de dispositivos

Neste momento, estão emparelhados dois dispositivos com o sistema CRIP: um *smartphone* e uma balança. Por isso, se esses dispositivos estiverem ativos nas imediações, o método `GetDevices` deverá conseguir detetar ambos os dispositivos. Para efeitos deste teste, foram retiradas as pilhas da balança `UC-321PBT-C`, de modo a que esta não fosse detetada. O resultado pode ser visto na figura 6.8.

```
[CripBluetooth.cpp::GetDevices:229] [GetDevices] Connected to C0:EE:FB:25:41:11 (OnePlus One)!
[CripBluetooth.cpp::GetDevices:239] [GetDevices] Could not connect to 00:09:1F:80:58:D1 (A&D WS UC-321PBT-C)!
[GetDevices] We tried every device we know! We managed to connect to 1 of them!
Device 1: OnePlus One
```

Figura 6.8: Teste ao método `GetDevices`.

Ou seja, o *smartphone* (que foi mantido com o Bluetooth em funcionamento) foi detetado, no entanto, a balança, como não está alimentada, não foi detetada, apesar de estar emparelhada com o sistema.

6.2.4 Recolha de dados médicos

Indo de encontro ao que é o objetivo principal do projeto *CareStore*, foi feita a recolha de dados médicos da balança `UC-321PBT-C`. Como já foi dito anteriormente, o método `ReadDevice` implementa um mecanismo de *timeout* que é ativado quando o dispositivo do qual se tenta recolher uma amostra não envia nenhuma amostra. Na figura 6.9 é possível verificar esse mecanismo em funcionamento.

```
[ReadDevices] Waiting for data from 00:09:1F:80:58:D1!

[CripBluetooth.cpp::ReadDevice:618] Timeout Reached!
[ReadDevices] Timeout Reached!
```

Figura 6.9: Mecanismo de *timeout* do método `ReadDevice`.

Após este exemplo, é importante agora apresentar um caso em que é recolhida uma amostra da balança. A figura 6.10 reflete essa situação.

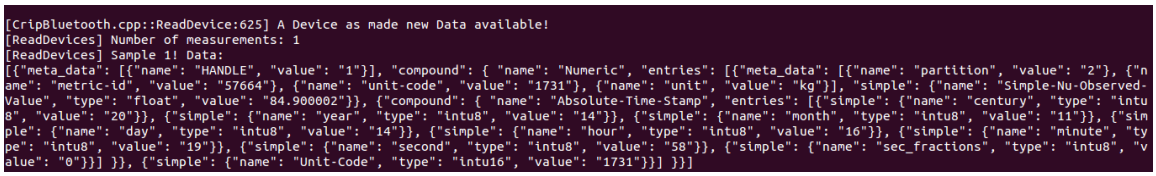


Figura 6.10: Teste ao método `ReadDevice`.

De modo a perceber melhor os dados enviados pela balança, foi retirada uma parte da string JSON, contendo o peso de uma amostra.

```
"simple":{
    "name":"Simple-Nu-Observed-Value",
    "type":"float",
    "value":"84.900002"
}
```

Capítulo 7

Conclusões e Trabalho Futuro

No final de todos os projetos, é importante fazer uma análise do trabalho desenvolvido de modo a perceber se os objetivos estabelecidos foram alcançados e quais as melhorias que se podem fazer ao projeto em questão.

No caso do projeto *CareStore*, os objetivos gerais do projeto ainda não foram completamente alcançados e há um longo caminho a percorrer de modo a garantir uma plataforma que cumpra todos os requisitos de funcionamento. No entanto, no que toca ao trabalho abrangido por este documento, ou seja, desenvolvimento de funcionalidades Bluetooth para o subsistema CRIP do *CareStore*, os objetivos foram cumpridos a 100%. Naturalmente, e como será ainda abordado ao longo deste capítulo, há melhorias que podem ser feitas ou alterações que poderiam beneficiar o projeto, no entanto, tendo em conta aqueles que eram os objetivos do projeto em termos de funcionalidades Bluetooth, estes foram cumpridos com sucesso.

Concluindo, olhando para o que foi criado no contexto desta dissertação, temos o seguinte:

- Módulo Bluetooth com API compacta e reutilizável, com suporte para as seguintes funcionalidades:
  - Descoberta e Emparelhamento de dispositivos Bluetooth;
  - Deteção de proximidade dispositivos Bluetooth ativos;
  - Leitura de parâmetros de saúde a partir dispositivos médicos com capacidade de comunicação Bluetooth;

- Criação de biblioteca de *software* para dar suporte a *tags* NFC, do tipo NTAG203;
- Criação de biblioteca de *software* para dar suporte a emparelhamento OOB via NFC, utilizando um esquema de *Static Handover*;
- Criação de biblioteca de *software* para invocação de métodos da API DBus da BlueZ 4;
- Criação de um agente de emparelhamento de dispositivos Bluetooth, adaptado às características de IO do CRIP;
- Preparação de SO para suporte das funcionalidades da plataforma CRIP.

Tendo em conta o produto final apresentado, é altura de falar um pouco das melhorias que ainda podem ser feitas relativamente às funcionalidades Bluetooth do projeto *CareStore*. Este está longe de ser um projeto acabado, e ainda serão feitas bastantes alterações de modo a tornar o sistema mais eficiente. Nesse sentido, são propostos os seguintes pontos, para trabalho futuro:

- Melhorias no controlo de acesso às funcionalidades Bluetooth do CRIP. Ou seja, neste momento, qualquer dispositivo se pode emparelhar com o CRIP. Poderá ser interessante implementar um sistema de admissão de dispositivos, baseado na sua *Class of Device*. Isto permite filtrar os dispositivos, de modo a apenas autorizar o emparelhamento de dispositivos relevantes para o CRIP (por exemplo, dispositivos médicos ou *smartphones*);
- Implementação do esquema de Negotiated Handover, de modo a tornar o esquema de emparelhamento OOB mais poderoso. A título de exemplo, seria interessante um utilizador usar as capacidades NFC do seu *smartphone* de modo a registar dispositivos médicos, utilizando uma aplicação desenvolvida para esse efeito. Um cenário de utilização seria o utilizador inserir os dados do dispositivo Bluetooth que pretendia emparelhar, utilizando uma aplicação especializada e disponibilizar esses dados ao CRIP através de NFC, fazendo com que este emparelhasse o dispositivo em questão;

- Aumentar o número de dispositivos médicos suportados pelo CRIP. Para esse fim, teriam de ser implementados os modelos de dados dos dispositivos que se pretendiam suportar, de acordo com as normas IEEE 11073 e ser feita a integração com o Antidote. Tendo em conta que neste momento apenas é suportada a comunicação com 4 tipos de dispositivos médicos, isto será, sem dúvida, um dos pontos a abordar no futuro, de modo a dar mais poder à plataforma CRIP;

- Outro dos focos principais do projeto será a migração da API para dar suporte à API da BlueZ 5, de modo a garantir que as funcionalidades Bluetooth seguem e implementam as especificações e normas mais recentes. Para além disto, esta versão da BlueZ apresenta melhorias significativas relativamente ao suporte ao Bluetooth *Low Energy*;

- O foco no Bluetooth *Low Energy* é precisamente um dos grandes focos do trabalho futuro. Na versão atual do CRIP, não é dado suporte de *software* a esta tecnologia, apesar do *hardware* suportar a comunicação *Low Energy*, devido ao funcionamento *dual mode*. Tendo em conta as vantagens em termos de consumo energético desta tecnologia, relativamente ao Bluetooth *Classic*, sem dúvida que dar suporte a dispositivos médicos *Low Energy* é um dos focos de maior importância para o futuro do projeto. A única razão para isto não ter acontecido até agora, é que a maioria dos dispositivos médicos ainda usa tecnologias Bluetooth *Classic*;

- Teste da plataforma CRIP no terreno. Ao mesmo tempo que este documento está a ser escrito, estão a ser feitos os primeiros testes em alguns lares dinamarqueses. No entanto este é um processo demorado, de modo a dar tempo aos utilizadores de utilizarem o sistema algumas vezes e recolher o seu *feedback*. Por isso, no futuro, estão previstos bastantes testes ao sistema.

Relativamente às outras componentes do projeto, estas também requerem trabalho futuro, no entanto isso está fora do âmbito deste documento, que se debruça apenas nas funcionalidades Bluetooth do projeto *CareStore*.

Bibliografia

[1] M. Memon, "CareStore: Platform for Seamless Deployment of Ambient Assisted Living Applications and Devices," May 14, 2014, http://pitlab.itu.dk/sites/default/files/CareStore%20Slides%20-%20PIT.pdf.

[2] Bluetooth SIG, "Types of Bluetooth Technologies," http://www.bluetooth.com/Pages/Bluetooth-Brand.aspx.

[3] ——, *Bluetooth Specification Version 4.0*, June 30, 2010, vol. 1, ch. Architecture and Terminology Overview.

[4] Adrio Communications Ltd, "NFC Modulation and RF Signal," http://www.radio-electronics.com/info/wireless/nfc/near-field-communications-modulation-rf-signal-interface.php.

[5] D. Santos, A. Perkusich, and H. Almeida, "Standard-based and Distributed Health Information Sharing for mHealth IoT Systems." 2nd IEEE International Workshop on Service Science for eHealth, 2014.

[6] I. Robert Bosch Healthcare Systems, *Health Buddy System*, 2014, http://www.bosch-healthcare.com/media/us/home/pbk_images/why_bosch_/F_03D_601_550_Rev04_HB_DS_2014.pdf.

[7] Google, *Google Fit: Platform Overview*, 2014, https://developers.google.com/fit/overview.

[8] S. Wagner, F. O. Hansen, C. F. Pedersen, M. Memon, F. H. Aysha, M. Mathissen, C. Nielsen, and O. L. Wesby, "CareStore Platform for Seamless Deployment of Ambient Assisted Living Applications and Devices." Pervasive Computing Technologies for Healthcare (PervasiveHealth), May 5, 2013.

[9] Bluegiga, "BT111 Bluetooth Smart Ready HCI Module Product Presentation," September, 2013, https://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/bt111-bluetooth--smart-ready-hci/documentation/.

[10] Bluetooth SIG, *Bluetooth Specification Version 4.0*, June 30, 2010, vol. 4, ch. Host Controller Interface.

[11] Red Hat, Inc., "D-Bus Tutorial," http://dbus.freedesktop.org/doc/dbus-tutorial.html.

[12] Nokia, "NFC Handover working principle," http://developer.nokia.com.

[13] ——, "Understanding NFC Data Exchange Format (NDEF) messages," http://developer.nokia.com.

[14] NFC Forum, *NFC Data Exchange Format (NDEF)*, July 24, 2006.

[15] ——, *Bluetooth Secure Simple Pairing Using NFC*, October 18, 2011.

[16] Bluetooth SIG, *Health Device Profile*, July 24, 2012.

[17] ——, *Bluetooth Specification Version 4.0*, June 30, 2010, vol. 3, ch. Core System Package.

[18] European Union, *The 2012 Ageing Report*, February, 2012, http://ec.europa.eu/economy_finance/publications/european_economy/2012/pdf/ee-2012-2_en.pdf.

[19] AAL Joint Programme, "AAL Objectives," http://www.aal-europe.eu/about/objectives/.

[20] The CareStore Consortium, "CareStore Project," http://www.carestore.eu/.

[21] S. Wagner, F. O. Hansen, C. F. Pedersen, M. Memon, F. H. Aysha, M. Mathissen, C. Nielsen, and O. L. Wesby, "Ambient Assisted Living Ecosystems of Personal Healthcare Systems, Applications, and Devices." Scandinavian Conference on Health Informatics, August 20, 2013.

[22] Bluetooth SIG, *Bluetooth Specification Version 4.0*, June 30, 2010, vol. 2, ch. BR/EDR Controller volume.

[23] ——, *Bluetooth Specification Version 4.0*, June 30, 2010, vol. 6, ch. Low Energy Controller volume.

[24] S. Wagner and C. Nielsen, "OpenCare project: An open, flexible and easily extendible infrastructure for pervasive healthcare assisted living solutions." 3rd International Conference on Pervasive Computing Technologies for Healthcare Pervasive Health, April 1, 2009.

[25] P. Wolf, A. Schmidt, and J. P. Otte, "OpenAAL - The Open Source Middleware for Ambient-Assisted Living (AAL)," 2010.

[26] M. R. Tazari, F. Furfari, Á. F. Valero, S. Hanke, O. Höftberger, and D. Kehagias, "The universAAL Reference Model for AAL," 2012.

[27] BlueZ, "BlueZ features," http://www.bluez.org/about/.

[28] BlueZ, *BlueZ 5 API introduction and porting guide*, 2012, http://www.bluez.org/bluez-5-api-introduction-and-porting-guide.

[29] Bluegiga, "BT111 Bluetooth Smart Ready HCI Module," https://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/bt111-bluetooth--smart-ready-hci/.

[30] Red Hat, Inc., "D-Bus Specification," http://dbus.freedesktop.org/doc/dbus-specification.html.