INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

ÁREA DEPARTAMENTAL DE ENGENHARIA DE ELETRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

Android as a Cloud Ticket Validator

Agostinho da Cunha Baía, nº 31589
(Licenciado em Engenharia Informática e de Computadores)

PROJETO PARA OBTENÇÃO DO GRAU DE MESTRE
EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Orientadores:
Professor Adjunto Doutor João Carlos Amaro Ferreira
Professor Adjunto Doutor Porfírio Pena Filipe

Júri:
Presidente:
Professor Adjunto Mestre Pedro Alexandre Seia Cunha Ribeiro Pereira (ISEL-ADEETC)
Vogais:
Mestre Gonçalo Vasco Trincão Bento da Cunha (*Link Consulting*)
Professor Adjunto Doutor João Carlos Amaro Ferreira (ISEL-ADEETC)
Professor Adjunto Doutor Porfírio Pena Filipe (ISEL-ADEETC)

JULHO DE 2013

Resumo

No âmbito do projeto foi desenvolvido um dispositivo, numa plataforma móvel, com o objetivo de implementar um validador de baixo custo para a bilhética de transportes coletivos de passageiros. O trabalho realizado é enquadrado no projeto SmartCITIES *Cloud Ticketing,* da Link Consulting SA, o qual propõe uma implementação *multi-tenancy* para terminais de bilhética. Adicionalmente, foi introduzido e aplicado o conceito de "*thin device*" que permite mover operações tradicionais de bilhética para uma nuvem computacional*,* permitindo desta forma aumentar a flexibilidade e interoperabilidade.

Assim, neste contexto, é proposta a migração dos terminais de validação tradicionais para uma nova abordagem implementada num *tablet* com o sistema operativo *Android*. Foram analisados diversos *tablets* com o objetivo de encontrar um dispositivo capaz de interagir com o leitor de cartões e lidar com os cartões eletrónicos Lisboa Viva (*smart card Calypso*), para verificar, numa fase inicial através de um serviço, se o contrato presente no título de transporte é válido. Baseado nesta aproximação flexível é possível criar outros dispositivos associados à bilhética (por exemplo, máquinas de venda, cancelas, entre outros) usando um processo de leitura comum e fazendo alterações apenas nos serviços *web*.

O dispositivo implementado pode ter a lógica de validação alojada numa nuvem computacional (*Amazon Web Services*). Fisicamente, o dispositivo é suportado num *tablet* com sistema operativo *Android* que acede a serviços alojados na nuvem. Parte deste resultado já se encontra em ambiente de produção na empresa Link.

Palavras-chave

Android, *Smart Cards*, Lisboa Viva, *Calypso*, Nuvem Computacional, Sistemas de Bilhética

Abstract

In this project work it was developed a productive validation device in a mobile platform, to achieve a low cost ticketing device public transport passenger. This work is integrated in the project SmartCITIES Cloud Ticketing from Link Consulting SA, used to propose a multi-tenancy implementation of ticketing terminals. Additionally, it was introduced the "thin device" concept which allows to move the traditional ticketing operations to cloud platform, allowing that way to increase elasticity and interoperability issues.

Therefore, in that context, it is proposed the migration of a traditional ticketing validator for a novel approach in a tablet device with Android operating system. Current work analyzed several tablets to choose one able to interact with the card reader and oriented to the *Lisboa Viva* electronic card (smart card Calypso) to check in a first phase the ticket contract in a service basis out of the ticketing device. Based on this flexible approach was possible to create other ticketing devices (e.g sale machines, gates devices and others) using the common read process and change only web services.

The device created is a prototype of this validation ticketing device with the validation services in a cloud platform (Amazon Web Services). Physically, the prototype consists in validator running on a tablet with Android operating system, but the validation logic remains out of the device in a service basis. Part of this result it's being used in production environment by Link.

Keywords

Android, Smart Cards, *Lisboa Viva*, Calypso, Cloud Computing, Electronic Ticketing Systems

Agradecimentos

Gostaria de agradecer a todas as pessoas que contribuíram direta ou indiretamente para a elaboração deste projeto.

Agradeço aos meus orientadores, Professor João Ferreira e Professor Porfírio Filipe, pelo apoio prestado, no decorrer da elaboração do projeto.

Agradeço também ao engenheiro Hugo Bicho (da empresa Link) pelo apoio prestado no desenvolvimento da biblioteca de interação com leitores e *smart cards* e ao engenheiro João Silva (da empresa Link) pelo apoio prestado no desenvolvimento do serviço de validação.

Agradeço ainda aos meus familiares e amigos, dos quais destaco os meus pais, pelo apoio que me deram.

Lista de acrónimos

AMI – *Amazon Machine Images*

APDU – *Application Protocol Data Unit*

API – *Application Programming Interface*

AWS – *Amazon Web Service*

CRC – *Cyclic Redundancy Code*

DES – *Data Encryption Standard*

DESX – *Data Encryption Standard X*

DOS – *Denial of Service*

Ec2 – *Amazon Elastic Compute Cloud*

EEPROM – *Electronic Erasable Programmable Read Only Memory*

EPC – *Electronic Product Code*

FIFO – *First In First Out*

HMAC – *Hash Message Authentication Code*

HTTP – *Hypertext Transfer Protocol*

HTTPS – *Hypertext Transfer Protocol Secure*

JBOSS AS7 – *JavaBeans Open Source Software Application Server 7*

JNI – *Java Native Interface*

JSON – *JavaScript Object Notation*

MAC – *Message Authentication Code*

NDK – *Native Development Kit*

NFC – *Near Field Communication*

OTLIS – Operadores de Transportes da Região de Lisboa

PC – *Personal Computer*

PIN – *Personal Identification Number*

QR - *Quick Response*

REST – *Representational State Transfer*

RFID – *Radio Frequency Identification*

S3 – *Amazon Simple Storage Service*

SaaS – *Software as a Service*

SAM – *Secure Application Module*

SFI – *Short File Identifier*

SO – Sistema Operativo

STIB – *Société des Transports Intercommunaux de Bruxelles*

TDES – *Triple Data Encryption Standard*

TI – Tecnologias de Informação

TID – *Tag Identifier*

TLD – *Type Length Data*

URL – *Uniform Resource Locator*

USB – *Universal Serial Bus*

WORM – *Write Once Read Many*

Índice

Índice de Figuras

1 Introdução

O presente trabalho enquadra-se na área dos sistemas de bilhética eletrónica e está associado a um projeto da Link Consulting SA [1] (Link) em parceria com o ISEL (Instituto Superior de Engenharia de Lisboa), designado por SmartCITIES *Cloud Ticketing* [2]. O objetivo geral do projeto SmartCITIES é desenvolver serviços integrados de bilhética de vasta abrangência funcional a custo reduzido.

O projeto SmartCITIES *Cloud Ticketing* propõe a evolução do modelo tradicional da bilhética para um modelo baseado numa nuvem computacional, em que os processos centrais da bilhética são oferecidos através de serviços numa lógica de SaaS (*Software-as-a-Service*), que podem ser subscritos pelos operadores de transporte e onde o pagamento do serviço é feito mediante a utilização.

De forma a atingir este objetivo genérico é necessário resolver os desafios levantados pela especificidade dos terminais da bilhética com aplicações embebidas, que não podem ser convertidas diretamente para tecnologia utilizável no contexto dos navegadores de *Internet* acedidos pelos utilizadores finais.

O projeto SmartCITIES *Cloud Ticketing* define uma arquitetura, que permite a construção de um *bus* de serviços comuns da bilhética ao qual os terminais se podem ligar de forma simples, idealmente num modelo *Plug-and-Play*, para que a nuvem computacional reconheça e configure automaticamente os equipamentos de bilhética no momento da instalação. Será por isso necessário definir a arquitetura do *bus* de serviços, assim como as características dos terminais para consumirem os serviços disponibilizados. Desta forma, atinge-se o objetivo final do projeto SmartCITIES, ou seja, a consolidação da lógica de negócio preferencialmente numa nuvem computacional.

Esta alteração de paradigma pretende beneficiar do facto dos serviços de bilhética na nuvem computacional poderem ser acedidos através da *Internet* tirando simultaneamente partido da elasticidade assegurada pela nuvem que oferece alta

disponibilidade, o que permite lidar com picos de utilização e com falhas que têm particular relevância em horas de ponta dos transportes públicos.

Desta forma toda a lógica aplicacional pode ser consolidada e implementada sobre protocolos abertos e seguros, tornando os equipamentos *frontend* mais simples, preferencialmente sem lógica local, beneficiando em simultâneo do facto de estar *online* com o sistema central do respetivo operador oferecendo funcionalidades de valor acrescentado e adequadas à utilização proposta.

A transferência da lógica aplicacional para a nuvem computacional permite ainda converter as aplicações de bilhética em aplicações abertas e independentes do fornecedor de *hardware*, permitindo uma maior flexibilidade na seleção do *hardware* mais adequado para cada aplicação específica.

Pretende-se que a solução de bilhética proposta, sediada na nuvem computacional, seja igualmente uma forma de promover a colaboração e interoperabilidade entre operadores de transporte principalmente em grandes áreas metropolitanas, que beneficiam do facto dos diferentes operadores estarem a utilizar serviços disponibilizados pela infraestrutura partilhada.

O alojamento na nuvem computacional também vai facilitar a adesão de pequenos operadores à bilhética eletrónica. Por exemplo, na indústria da aviação já vemos sistemas de reserva de lugares e de venda de bilhetes a serem oferecidos "como serviço" para várias companhias aéreas, muitas vezes a um custo de apenas alguns cêntimos por bilhete. Na verdade, poucos operadores de *low cost* administram e mantêm o seu próprio sistema de bilhética porque as opções SaaS disponíveis no mercado são uma alternativa mais eficiente e a custo mais reduzido.

Neste projeto, grande parte do esforço está em conseguir colocar ou integrar os equipamentos tradicionais de bilhética (i.e. validadores de cartões, máquinas de venda de bilhetes, máquinas de controlo de acessos entre outros) na nuvem computacional, passando de um conceito de "*fat devices*" para "*thin devices*". Na Figura 1 podem observar-se três exemplos de "*fat devices*", que são uma cancela automática, um validador e uma máquina de venda automática, respetivamente. No modelo "*fat*

device" cada equipamento é construído de forma integrada para executar uma determinada função, não sendo permitida muitas vezes a integração de diversos equipamentos de fabricantes diferentes.

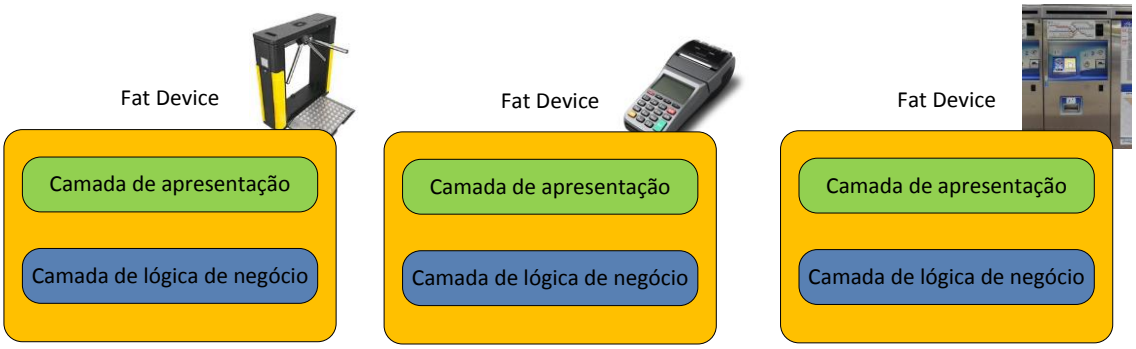

Figura 1 – Exemplo de três arquiteturas "*fat device*".

Pretende-se desenvolver uma nova aproximação para este tipo de equipamento, designada por "*thin device*" na qual a camada da lógica do negócio passará a estar na nuvem computacional, permitindo assim a interoperabilidade entre funções e equipamentos (ver Figura 2).

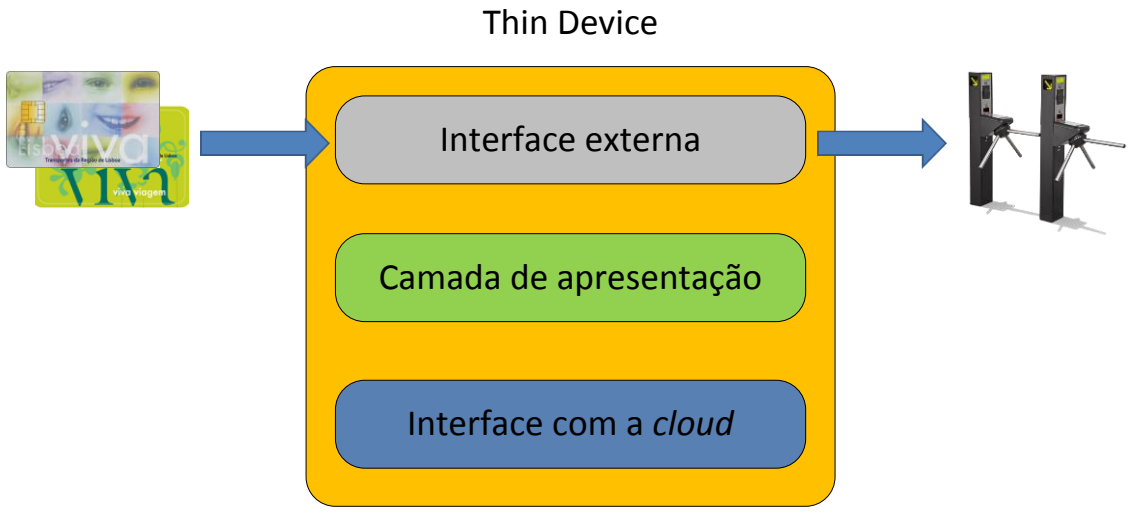

Figura 2 – Exemplo de arquitetura "*thin device*".

Na Figura 3 podem observa-se três dispositivos, que podem ser considerados como "*thin devices*". O primeiro dispositivo é uma máquina de venda automática, que permite a um utilizador adquirir títulos de transporte, ou carregar títulos previamente adquiridos.

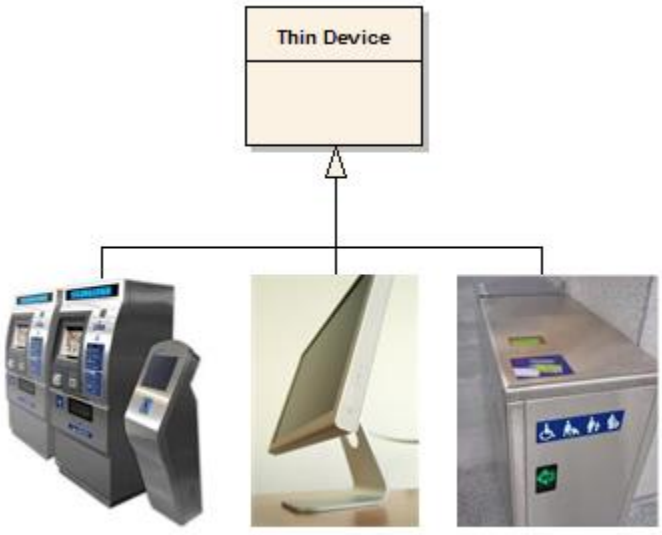

Figura 3 – Exemplo de três "*thin devices*"

O segundo dispositivo apresentado na Figura 3 é um dispositivo utilizado nos pontos de venda (*point of sale*) e é utilizado por um funcionário para realizar venda e carregamento de títulos de transporte. Este dispositivo pode ser um computador, que se encontra ligado a um leitor de cartões, e que é utilizado para interagir com um *smart card* podendo por exemplo carregar o saldo ou incrementar o número de unidades disponíveis no cartão.

O último dispositivo presente na Figura 3 representa uma cancela, este dispositivo para além de realizar as mesmas funcionalidades que um validador, tem ainda como funcionalidade permitir a entrada do passageiro apenas se o seu título de transporte for válido. Desta forma quando o passageiro apresenta um cartão com unidades ou saldo suficiente a cancela é aberta, sendo posteriormente decrementado o número de unidades, ou removido do saldo o valor da viagem. Nestes três dispositivos que poderão ser "*thin devices*" a lógica de negócio, que permite que seja por exemplo aberta uma cancela, encontra-se armazenada na nuvem computacional.

1.1 Arquitetura do Projeto SmartCITIES *Cloud Ticketing Services*

A arquitetura do sistema de bilhética eletrónica proposta no projeto SmartCITIES, está dividida em vários serviços tal como se pode verificar na Figura 4 [2]. Cada serviço funciona como se fosse uma "camada" e pode ser acedido diretamente por dispositivos e/ou por outros serviços.

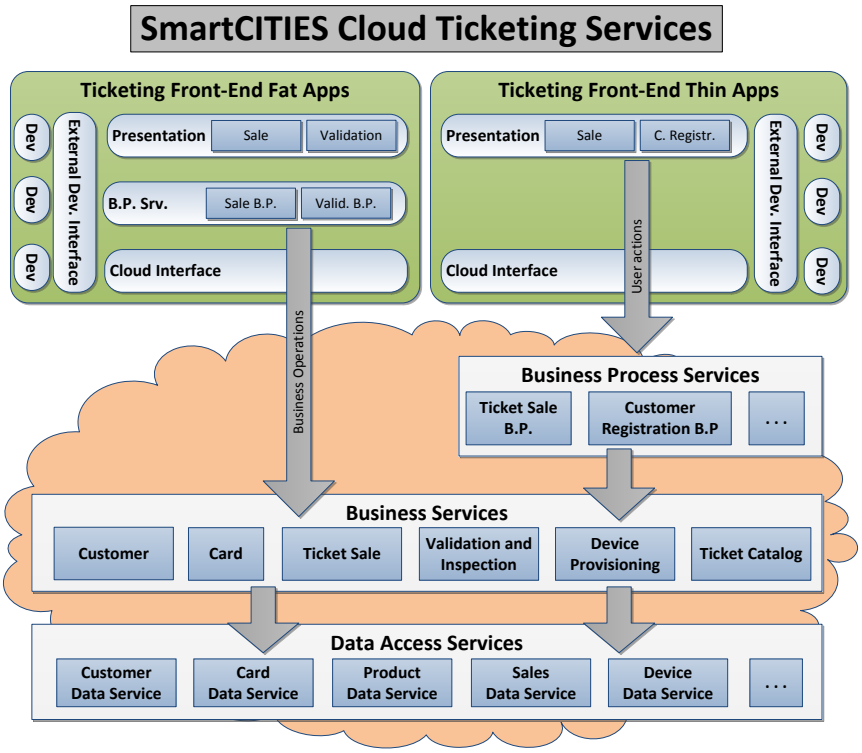

Figura 4 – Arquitetura proposta para *cloud ticketing* [2].

A ideia de transferir uma maior responsabilidade para a nuvem computacional tem como objetivo facilitar a integração de vários operadores de transportes diferentes, de vários pontos de venda e com um vasto número de dispositivos que poderão ser utilizados para funções como verificar se um título de transporte é válido ou ainda ser utilizados para realizar operações de *back office*.

Esta integração é possível utilizando um conjunto de serviços alojados na nuvem computacional que seriam comuns a todos os operadores e dispositivos.

Na Figura 4, é possível observar-se os dois tipos de dispositivos, "*fat device*" e "*thin device*" e a sua responsabilidade na arquitetura, que está dividida em três camadas. A primeira camada é a de apresentação e está presente nos dois tipos de dispositivos. A segunda camada contém a lógica de negócio, e está apenas presente nos dispositivos

do tipo "*fat device*" e interage diretamente com a camada *Business Services*, que se encontra na nuvem computacional. Por sua vez os dispositivos do tipo "*thin device*" não têm lógica de negócio local, pois a lógica de negócio que estes dipositivos utilizam encontra-se armazenada na nuvem computacional e é disponibilizada pelos *Business Process Services*. A última camada permite o acesso a dados, e está disponível através dos *Data Access Services* que estão presentes na nuvem computacional.

Esta arquitetura, pode ser utilizada em simultâneo, tanto por "*thin devices"*, como por *"fat devices"*, o que permite que continuem a ser utilizados dispositivos do tipo "*fat device*", em condições, onde eventualmente não possam ser utilizados dispositivos do tipo "*thin device"*.

Os *Data Access Services* permitem que sejam acedidos e manipulados os dados onde estão por exemplo as informações sobre os bilhetes vendidos, ou ver informações sobre um dado cliente.

Os *Business Services* permitem realizar algumas operações como listar o catálogo de títulos existentes, carregar um título ou ainda inserir os dados de um novo cliente.

Os *Business Process Services* coordenam a utilização entre vários *Business Services* para implementar casos de utilização.

1.2 Objetivo

O presente trabalho enquadra-se na área da bilhética eletrónica, e tem como objetivo desenvolver um terminal de bilhética de baixo custo associado ao cartão Lisboa Viva. O trabalho desenvolvido permite validar o conceito de "*thin device*" proposto no projeto SmartCITIES *Cloud Ticketing*, identificada na Figura 2, por forma a permitir integrar no mesmo equipamento diversas interfaces como seja a de um leitor de cartões ou a comunicação com um serviço *web* onde é realizada a validação de títulos de transporte.

Para o efeito o presente trabalho pretende desenvolver numa plataforma inovadora (*tablet* com SO (Sistema Operativo) *Android*) um dispositivo de leitura de cartões (*smart cards Calypso)*, o qual permite verificar a validade dos títulos de transporte

existentes nesses *smart cards*, estando separado o processo de leitura e de validação do bilhete, como indica a Figura 2.

Atualmente os *smart cards*, são utilizados quer em passes, quer em bilhetes e são carregados eletronicamente. O novo terminal implementado no âmbito deste projeto, é um *tablet* com SO *Android* que interage com um leitor de cartões, o que permite realizar as operações necessárias sobre *smart cards*, para verificar se um título está válido no momento. Esta solução tem também como característica o facto da lógica referente ao processo de validação estar fora do dispositivo numa lógica de serviços para poder permitir a reutilização de partes do processo, visto que algumas partes são semelhantes em diversos operadores.

A Figura 5 ilustra os três componentes que estão envolvidos no trabalho desenvolvido no âmbito deste projeto, que são a biblioteca que permite interagir com o leitor e com os *smart cards*, o validador *Android*[1] e o serviço de validação. Cada um destes componentes é abordado nos capítulos 4, 5 e 6 respetivamente.

Figura 5 – Componentes envolvidos no projeto desenvolvido.

Inicialmente foi analisado um "*fat device*", que consistia num computador que utiliza um leitor ASK [3] para interagir com *smart cards*. De seguida foi construída uma biblioteca em *Java* para correr num *tablet* com SO *Android* e que desempenha as mesmas funcionalidades que o código que corre no computador (realizado na linguagem C). No entanto este *tablet* com SO *Android* é um "*thin device*", que utiliza a lógica de validação presente num serviço *web*. É de salientar que a escolha do SO *Android* foi imposição da empresa Link, a qual identificou como oportunidade de negócio a existência destes dispositivos. Esta escolha por parte da empresa Link deve-se ao facto do SO *Android* ser um dos SO líderes de mercado e de os *tablets* com SO

[1] Aplicação de validação desenvolvida para o *tablet* com SO *Android* e utilizada para validação de contratos presentes em *smart cards.*

Android terem um custo reduzido (cerca de 250 euros), em comparação com os terminais existentes atualmente (acima dos 1000 euros).

O *tablet* com SO *Android* utilizado no contexto deste projeto terá de ser escolhido mediante as condições necessárias para a utilização na área da bilhética como dispositivo validador.

Ao contrário dos validadores "*fat device*" que são feitos na linguagem C, o validador *Android* foi desenvolvido totalmente na linguagem *Java*.

1.3 Organização do Projeto

Este documento é dividido em oito capítulos, e ainda um capítulo de referências. Os capítulos apresentados são dirigidos para diferentes níveis de conhecimento. Esta divisão destina-se a que o documento seja percetível para qualquer pessoa, desde que tenha um nível de conhecimento tecnológico básico. O grau de conhecimento necessário para compreender os diversos capítulos cresce de forma gradual ao longo do documento.

O capítulo 2 é o capítulo tradicional do estado da arte com a particularidade de ir contribuir com um *deliver* do projeto SmartCITIES:

- Estado da arte dos sistemas de bilhética eletrónicos, RFID (*Radio Frequency Identification*) e *smart cards.*

No capítulo 6 é fornecida uma visão sobre soluções e tecnologias que possam ser utilizadas no campo da bilhética eletrónica, como as etiquetas RFID e *smart cards*. São também abordadas as formas de comunicar com um *smart card* e ainda aspetos de segurança relacionados com a bilhética e com os *smart cards*.

Figura 6 – Arquitetura da solução.

Os capítulos 3 a 6, estão relacionados com o trabalho desenvolvido para atingir os objetivos propostos e foram organizados de acordo com as camadas apresentadas na Figura 2 e com a arquitetura ilustrada pela Figura 6, assim: capítulo 3 – *Android as a Cloud Ticket Validator*, onde é descrito de forma genérica quais são os componentes do projeto desenvolvido e como foi feita a escolha do leitor de cartões e do *tablet com SO Android*; capítulo 4 – Interface Externa de Leitura de *Smart Cards*, é descrito o processo de leitura de *smart cards* implementado; capítulo 5 – Validador desenvolvido para o SO *Android*, é descrito o processo de interação com a interface externa e a nuvem computacional por forma a criar um dispositivo de validação de bilhetes usando o conceito de "*thin device"*; capítulo 6 – Camada de Lógica de Validação, é abordado o processo de interação entre o *tablet* com SO *Android* e serviço de validação por forma a verificar se o contrato lido do cartão Lisboa Viva é ou não válido.

Os aspetos de segurança enunciados nos capítulos 2 e 3 não foram utilizados no desenvolvimento do projeto, para não sobrepor o trabalho do projeto corrente ao trabalho do projeto "SmartCards" [4] que se vai focar na utilização deste módulo de segurança.

No capítulo 7 é descrito como os várias componentes desenvolvidas no âmbito deste projeto podem ser utilizados em diferentes cenários reais, indicando se seria ou não necessário fazer alterações e em caso afirmativo quais as alterações necessárias.

No capítulo 8 é abordada a avaliação de resultados, e são descritos os testes feitos ao sistema. É também explicada a adequação dos testes ao sistema em causa e em consequência são ainda analisados e explicados os resultados dos mesmos.

Finalmente no capítulo 9 é apresentada a conclusão e o trabalho futuro.

2 Sistemas Eletrónicos de Bilhética

Neste capítulo é apresentado o estado da arte relativo aos sistemas eletrónicos de bilhética de forma a criar contexto para o trabalho desenvolvido. O conteúdo deste capítulo foi usado como base para o desenvolvimento do estado da arte do projeto SmartCITIES na área dos sistemas de bilhética eletrónicos, RFID (*Radio Frequency Identification*) e *smart cards*.

Na área dos transportes é importante que a secção de bilhética seja eficiente e segura, permitindo desta forma, por exemplo, controlar mais facilmente a validade de um título de transporte ou ainda aumentando a reutilização do suporte[2], visto que o mesmo título pode ser recarregado várias vezes.

Muitos dos títulos de transporte eletrónicos utilizam a tecnologia RFID ou *smart cards* para comunicar com os restantes dipositivos tecnológicos associados à bilhética eletrónica (por exemplo: validadores, cancelas automáticas, pontos de venda automáticos, entre outros).

Para definir um sistema de bilhética eletrónica é necessário ter em conta um conjunto de especificidades que cada empresa (ou conjunto de empresas) da área dos transportes tem. Ou seja, pode ser necessário implementar um sistema onde existam várias topologias de funcionamento, por exemplo: considerar que o custo de uma viagem permite que o utilizador circule dentro de uma zona ou que o custo de uma viagem permite viajar em várias zonas durante um determinado tempo, sem custo acrescido para o passageiro.

É necessário também ter em conta o tipo de transporte, por exemplo: um transporte do tipo expresso pode ter necessidades, no âmbito da bilhética, diferentes de um transporte do tipo urbano. No caso de uma viagem expresso tipicamente o passageiro compra uma viagem, ou uma viagem de ida e volta, sendo que o custo da viagem pode variar de acordo com a distância percorrida. Por sua vez, no caso de viagens urbanas

[2] Um suporte pode ser, por exemplo, um cartão com um *smart card* ou um telemóvel com NFC.

um passageiro pode com o mesmo título de transporte circular numa região sem qualquer custo acrescentado.

A própria emissão de títulos pode variar, sendo possível, por exemplo, que um título seja cobrado por viagem, por tempo ou ainda por zonas.

2.1 Validação *Offline* e *Online*

Existem dois tipos de validação que podem ser utilizados para verificar se a informação presente num cartão é ou não válida (validação *online* e validação *offline*). Na área da bilhética podem ser utilizadas as duas formas de validação.

Na validação *online* é utilizado um serviço central onde a informação de validação está armazenada, sendo que nos títulos de transporte apenas está um identificador para a informação referente ao título. Desta forma, um dispositivo validador necessita de comunicar com o serviço central, para saber se um título de transporte é ou não válido. Uma vantagem associada à validação *online* é o facto de o título de transporte poder ser mais simples, não necessitando em alguns casos de uma grande capacidade de processamento, uma vez que não é realizado qualquer tipo de operação criptográfica. No entanto este tipo de validação tem a desvantagem de obrigar que todos os validadores estejam constantemente ligados ao serviço central para que possam validar títulos. Esta técnica de validação é geralmente utilizada por títulos que contêm etiquetas RFID.

A validação *offline* permite que um título seja validado sem que o validador tenha necessidade de comunicar com um serviço central. Desta forma é necessário que o validador consiga apenas através da informação presente no cartão detetar se o título é ou não válido. Ou seja, é necessário que um cartão tenha a informação que permite validar o seu título de forma *offline*. Assim é necessário garantir que o título contém apenas informação legítima, não permitindo que se consiga forjar um título escrevendo informação no cartão. Este tipo de cartões, são geralmente *smart cards* com capacidade de processamento e que utilizam algoritmos criptográficos para garantir que a informação, neles armazenada, é autêntica. A vantagem deste tipo de validação consiste em permitir a validação de um título apenas com a informação

presente no respetivo cartão, ou seja, não é necessário utilizar um serviço central para o processo de validação. No entanto tem como desvantagem a existência de uma dificuldade acrescida na desativação de um título, por exemplo, no caso de roubo. Isto acontece porque para a validação é apenas tido em conta a informação presente no cartão.

Apesar de no contexto desta projeto se utilizar *smart cards* que possibilitam a validação *offline*, é no entanto utilizado um serviço de validação para verificar se o título de transporte é válido, desta forma a lógica de validação estará apenas no servidor de validação e não em cada dispositivo.

2.2 RFID (*Radio Frequency Identification*)

A RFID é uma tecnologia sem fios que permite comunicação e identificação, sendo por isso usada em alguns títulos de transporte, permitindo desta forma o desenvolvimento de bilhetes ou passes eletrónicos sem contacto. Cada um destes bilhetes eletrónicos tem uma etiqueta RFID que comunica com um leitor enviando a informação necessária para a identificação e validação do título de transporte.

É possível agrupar as várias etiquetas existentes em grupos, classificando-as através das suas diferentes características.

Uma forma de classificar as etiquetas pode ser em relação à sua forma de alimentação, existindo assim três tipos de etiquetas: as etiquetas ativas, as etiquetas passivas e as etiquetas semi-passivas [5].

Nos próximos subcapítulos são abordadas algumas características presentes nas etiquetas RFID.

2.2.1 Etiquetas Ativas

As etiquetas ativas contêm alimentação própria, ou seja, cada etiqueta é alimentada por uma bateria. A utilização da bateria tem como vantagens o facto de a distância entre a etiqueta e o leitor poder ser grande e ainda o facto de a etiqueta poder iniciar a comunicação com o leitor por iniciativa própria.

Este tipo de etiquetas tem como desvantagem o facto de ser necessário recarregar ou substituir a bateria, o que aumenta o seu custo. Tem ainda como desvantagem o facto de o tamanho ser maior que os outros tipos de etiquetas, devido incorporar a bateria.

2.2.2 Etiquetas Passivas

As etiquetas passivas, são etiquetas que não precisam de alimentação interna para comunicar com o leitor. Para funcionar as etiquetas passivas utilizam uma técnica denominada de *backscatter*, sendo que com esta técnica as etiquetas são ativadas quando recebem energia suficiente, presente no campo eletromagnético emitido pelo leitor.

O tempo de vida destas etiquetas é maior que o tempo de vida das etiquetas ativas pois não é necessário utilizar bateria. Este facto também contribui para que seja reduzido o custo de produção deste tipo de etiquetas.

A desvantagem deste tipo de etiquetas prende-se com o facto de apenas ser possível iniciar uma comunicação quando a etiqueta se encontra dentro do campo magnético emitido pelo leitor.

2.2.3 Etiquetas Semi-Passivas

As etiquetas semi-passivas utilizam bateria tal como as etiquetas ativas, no entanto só iniciam o seu funcionamento quando se encontram perto de um leitor tal como as etiquetas passivas. Desta forma as etiquetas passivas têm um tempo de vida maior, pois não estão constantemente a consumir bateria.

No entanto a comunicação com o leitor só pode ser iniciada quando a etiqueta está próxima do leitor, ou seja, apesar da existência de bateria estas etiquetas não têm o mesmo limite de distância, para comunicar com o leitor, que as etiquetas ativas.

2.2.4 Arquitetura de Memória do RFID

A arquitetura de memória de uma etiqueta RFID inclui quatro áreas de memória [6]:

- Memória reservada 1 – esta memória contém uma *password* que permite o acesso e a utilização da etiqueta e ainda uma *password* de *kill* que permite desativar o dispositivo, impedindo totalmente futuras utilizações;

- Memória reservada 2 – esta memória contém um CRC (*Cyclic Redundancy Code*) de 16 bits, um protocolo de controlo e ainda um código único que identifica a etiqueta: EPC (*Electronic Product Code*);
- Memória TID (*Tag Identifier*) – memória utilizada para informar o leitor RFID das funções opcionais que a etiqueta pode suportar, juntamente com dados referentes ao fabricante;
- Memória *user defined* – memória que o utilizador pode utilizar para armazenar os seus dados.

2.2.5 Tipos de Memória para Guardar Dados

As etiquetas RFID podem ter dados guardados, que poderão posteriormente ser enviados para o leitor. A memória de dados pode ser de um dos seguintes três tipos [7]:

1. *Read Only* – a informação presente nestas etiquetas é imutável e foi inserida durante o processo de fabrico;
2. WORM (*Write Once Read Many*) – neste tipo de memória é apenas possível escrever na memória uma única vez. A informação escrita pode ser lida mais que uma vez;
3. *Read-Write* – este tipo de memória permite várias leituras e escritas.

2.3 Segurança na Utilização de RFID

Como já foi dito anteriormente, o RFID é uma tecnologia sem fios, e como qualquer tecnologia sem fios podem ser levantados problemas de segurança, como ataques à disponibilidade ou ataques que ponham em causa a privacidade de um portador de um etiqueta RFID [8].

Nos próximos subcapítulos são enumerados alguns problemas de segurança.

2.3.1 Falsificação de Títulos

Um dos problemas associados a títulos de transportes é a possibilidade de falsificação dos mesmos, permitindo desta forma que alguém utilize os transportes sem que tenha pago para isso. Esta falsificação pode ser realizada clonando a etiqueta, utilizando por exemplo um ataque de *man-in-the-middle*, em que o atacante guarda a informação que é trocada entre a etiqueta de um passageiro legítimo e o leitor RFID da empresa

de transportes, podendo mais tarde utilizar a informação obtida para se fazer passar pelo utilizador legítimo [8].

2.3.2 Privacidade

As etiquetas RFID são utilizadas em títulos de transporte e como tal interagem com um leitor para identificar o título e verificar se o mesmo se encontra válido. Pode-se então colocar um problema de privacidade, caso seja possível identificar os locais onde o passageiro entra e sai de um meio de transporte. Deve-se então garantir que esta informação não seja acedida por entidades que não sejam de confiança.

2.3.3 Disponibilidade

A disponibilidade é um dos requisitos necessários num sistema informático, sendo desta forma necessário prevenir ataques, que diminuam a disponibilidade. Um dos ataques pode consistir num ataque DOS (*Denial of Service*), onde os utilizadores legítimos deixam de poder utilizar um transporte.

Outro ataque contra a disponibilidade pode consistir em adulterar os títulos de transporte de utilizadores legítimos, fazendo desta forma com que estes sejam considerados inválidos pelos validadores. Este ataque pode ser realizado escrevendo informação no *smart card* presente no título de transporte do utilizador, apagando ou danificando a informação colocada por quem emitiu o título de transporte.

2.3.4 Requisitos de Segurança

Para prevenir ataques à disponibilidade dos serviços de bilhética, à privacidade dos utilizadores e impedir a utilização de títulos forjados, devem-se manter nos sistemas intervenientes na bilhética eletrónica mecanismos que garantam a autenticidade e a privacidade [8].

2.3.4.1 Autenticação

É necessário garantir a autenticação permitindo que apenas os utilizadores legítimos conseguem utilizar o serviço. Os cartões não devem ser forjáveis nem deve ser possível "copiar/clonar" um título de transporte válido. Mesmo que seja possível ler sem restrições a informação presente num cartão, essa informação não deve ser

considerada autêntica se for escrita num outro cartão, por alguém não autorizado a emitir títulos.

Deve também ser garantida a viabilidade para que os utilizadores legítimos possam usar o sistema.

2.3.4.2 Privacidade

Os títulos de transporte eletrónicos permitem que seja facilmente recolhida informação sobre os trajetos em que foram utilizados, se o título de transporte estiver associado à identidade de um utilizador, podem existir problemas de violação da privacidade.

Deve então ser garantida a confidencialidade de quem utiliza o serviço, não permitindo o acesso a dados confidenciais, presentes nos *smart cards*, ou nos servidores. Deve ser garantido o anonimato de quem utiliza o título de transporte e não deve ser possível saber qual o percurso que um utilizador realizou.

2.4 Eficiência e Escalabilidade

Dois requisitos essenciais nos títulos de transporte eletrónicos são a eficiência e a escalabilidade, ou seja, um sistema que suporte este tipo bilhetes deve suportar um grande número de pedidos de validação, e a validação deve ser feita rapidamente, para permitir por exemplo, que um utilizador valide o seu título sem que seja necessário parar de andar para que uma cancela se abra [8].

Garantir a eficiência pode ser difícil, quando se utilizam *smart cards* de baixo custo, onde as respetivas capacidades de armazenamento e processamento são reduzidas, não sendo estes dispositivos capazes de realizar operações criptográficas.

2.5 *Smart Cards*

Um *smart card* pode ser definido como um circuito integrado embebido num cartão. Existem três tipos de circuitos: *memory-only*, *wired logic* e *secure microcontroller* [9].

Os cartões do tipo *memory-only* conseguem armazenar até 16 kilobits e têm um "custo" de leitura e escrita mais reduzido que os cartões magnéticos. No entanto estes cartões não têm capacidade de processamento para realizar cálculos, ou seja, estes

cartões servem apenas para armazenar dados. Alguma da informação presente nestes cartões pode estar protegida por um código PIN (*Personal Identification Number*) sendo que apenas é possível aceder a essa informação após a introdução do respetivo código PIN.

Os cartões *wired logic* permitem que o conteúdo presente na memória do cartão seja cifrado e autenticado. Estes cartões disponibilizam um sistema de ficheiros estático que suporta múltiplas aplicações, no entanto os comandos que este cartão aceita apenas podem ser alterados se a lógica do mesmo cartão for redesenhada.

Os cartões *secure microcontroller* contêm um microcontrolador e um sistema operativo, juntamente com a memória, o que permite que este cartão realize cálculos, por exemplo, para operações criptográficas. Este tipo de cartão é o cartão vulgarmente conhecido por *smart card*.

2.5.1 Interfaces de Comunicação

Um *smart card* necessita de comunicar com um leitor de cartões para que possa ser utilizado. A comunicação pode ser realizada de duas formas (com contacto e sem contacto), cada uma com uma interface dedicada [10].

Na interface de comunicação com contacto é necessário inserir o cartão num leitor, desta forma o leitor fica em contacto com um micro modulo condutor existente na superfície do *smart card*.

Por sua vez na interface sem contacto basta aproximar o cartão do leitor para que se inicie a comunicação. Esta comunicação é iniciada assim que o *smart card* detete que está próximo de um leitor.

Um *smart card* pode ser compatível com as duas interfaces de comunicação.

2.5.2 Memória

Os dados armazenados nos *smart cards* são guardados utilizando memória do tipo EEPROM (*Electronic Erasable Programmable Read Only Memory*). Este tipo de memória tem custos elevados, o que juntando ao facto de o *smart card* ter um

tamanho reduzido, faz com que a quantidade de memória presente nestes dispositivos seja também reduzida [9].

2.6 *Smart Cards Calypso*

Um dos tipos de *smart cards* mais utilizados na área da bilhética são os *smart cards Calypso*. Um exemplo da utilização destes *smart cards* são os passes Lisboa Viva utilizados nos transportes públicos da área de Lisboa.

Pode-se definir a tecnologia *Calypso* como sendo um conjunto de especificações técnicas que permitem uma transação segura e rápida entre um "objeto portátil" e um leitor [10].

Para além dos cartões dedicados utilizados nos títulos de transporte, um objeto portátil pode ser um dispositivo qualquer que implemente o conjunto de especificações técnicas *Calypso*, como por exemplo um telemóvel com tecnologia NFC (*Near Field Communication*).

Uma particularidade nos *smart cards Calypso* prende-se com o facto de todas as operações poderem ser realizadas de forma "transacional", o que assegura que os dados presentes no cartão estão sempre num estado consistente.

2.6.1 Segurança na Utilização de *Smart Cards Calypso*

A segurança nos *smart cards Calypso* é garantida utilizando cifra simétrica com algoritmos como o DESX (*Data Encryption Standard X*) ou o TDES (*Triple Data Encryption Standard*), juntamente algoritmos como MAC (*Message Authentication Code*), através do componente SAM (*Secure Application Module*) [10] e [11].

O SAM é também um *smart card* utilizado do lado do leitor para garantir a autenticação tanto do terminal de leitura, como do *smart card* presente no título de transporte.

Como os *smart cards* têm uma capacidade computacional reduzida, são utilizados algoritmos de cifra simétrica em detrimento de algoritmos de cifra assimétrica, por questões de desempenho.

Cada cartão tem três chaves simétricas, cada uma para uma função diferente: personalização, carregamento e débito. Cada uma destas chaves resulta de um processo diversificação entre o número de série do *smart card* e uma chave principal (*master key*) presente no SAM.

No entanto, para garantir que a chave simétrica utilizada é única para cada *smart card*, a mesma é gerada utilizando um processo de diferenciação entre o número de série do próprio *smart card* e a chave mestra (*master key*) presente no SAM, que é comum a todos os SAM.

Desta forma apesar de estar a ser utilizado um mecanismo de cifra simétrica, se uma das chaves presentes num dos cartões for "descoberta", apenas um dos *smart cards* será afetado, sendo por isso possível resolver o problema colocando esse *smart card* numa "lista negra", passando a partir desse momento a ser ignorado.

Desta forma é possível utilizar um mecanismo de cifra simétrica garantindo um desempenho superior comparativamente com a utilização de um mecanismo de cifra assimétrica.

A comunicação entre o *smart card* e o terminal, pode decorrer de forma a que ambos se autentiquem para garantir, por exemplo, que não ocorreu falsificação do título de transporte.

2.6.2 Chaves Utilizadas pelo *Smart Card Calypso*

O *smart card Calypso* utiliza três chaves simétricas: uma para operações de débito, outra para operações de carregamento e terceira é utilizada pelo emissor. Existe uma hierarquia entre as três chaves presentes no cartão, permitindo que todas as operações que podem ser realizadas pela chave de débito possam ser utilizadas pelas chaves de carregamento e pela chave do emissor. Todas as operações que podem ser realizadas com a chave de carregamento podem ser realizadas com a chave do emissor.

Não é possível forjar nenhuma das chaves que estão guardadas no *smart card Calypso*, pois as chaves nunca são transmitidas para o exterior. Desta forma é possível garantir a autenticidade dos dados presentes no *smart cards* do título de transporte.

2.6.3 Sessão Segura com o *Smart Card Calypso*

Para comunicar com o *smart card* de forma a garantir a autenticidade dos dados lidos ou escritos, é necessário executar as operações sobre o *smart card* dentro de uma sessão segura. Uma sessão garante a autenticidade do terminal, do *smart card* e da informação trocada entre o *smart card* e o terminal. Numa sessão é garantida também a consistência dos dados armazenados no cartão.

Do lado do leitor é necessário utilizar um SAM (*Secure Application Module*) que consiste também num *smart card*, e é responsável por realizar as operações seguras. É no SAM que está armazenada a *master key*, que é utilizada para autenticar a informação trocada entre o *smart card* e o leitor.

Como se pode observar na Figura 7 para iniciar uma sessão segura o terminal de leitura envia o comando *open secure session*. No entanto para criar uma sessão segura o SAM presente no lado do leitor, necessita de saber qual a chave simétrica que deve utilizar. A chave simétrica em causa é diferente para todos os cartões e é derivada através de um processo de diversificação entre o número de série do *smart card* e a *master key* do SAM. Desta forma o SAM precisa de saber qual o número de série do *smart card* para calcular a chave simétrica em causa e estabelecer uma sessão segura. Daí que antes de estabelecer uma sessão segura seja necessário utilizar o comando *select application* (descrito no capítulo 4.7.5) para obter o número de série do *smart card*.

Após o comando *open secure session* ser executado com sucesso é iniciada uma sessão segura entre o *smart card* e o leitor. A partir deste momento é possível ao leitor enviar comandos que manipulam dados presentes no cartão.

Após as alterações aos dados serem efetuadas é necessário que o leitor envie um comando para terminar a sessão. É nesta altura que é feita a autenticação da informação trocada, juntamente com a autenticação dos intervenientes (*smart card* e terminal). Para a autenticação é calculado um MAC de 8 bytes sobre todos os comandos trocados e ainda sobre dois números aleatórios, cada um gerado por um dos intervenientes na transação. Este MAC é obtido tanto pelo terminal como pelo

leitor e para iniciar o processo de fecho de sessão o terminal envia para o cartão os primeiros 4 bytes do MAC, e o cartão envia o terminal os últimos 4 bytes. Se o MAC final coincidir dos dois lados, significa que ambos se autenticaram corretamente e que o cartão conseguiu escrever os dados sem erros.

Caso o leitor envie um comando que resulte num erro, ou não envie o comando de fecho de sessão (por exemplo: devido a uma falha de comunicação por o *smart card* já não estar junto ao leitor), todos os comandos enviados para o *smart card* dentro da sessão corrente são revertidos (*rollback*). Desta forma é garantida a autenticidade e consistência dos dados presentes, ou seja, ou todos os comandos se executam com sucesso, ou nenhum comando é executado com sucesso. Caso a autenticação do leitor falhe os comandos enviados pelo leitor são também revertidos garantido desta forma que não é possível, por exemplo, forjar um título de transporte legítimo.

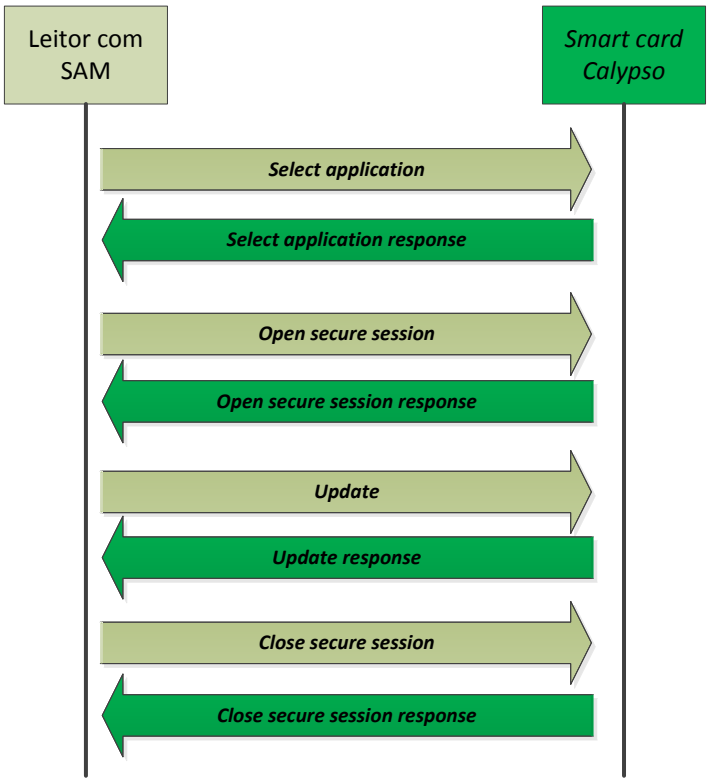

Figura 7 – Exemplo de comandos enviados numa sessão segura com o *smart card*.

3 *Android as a Cloud Ticket Validator*

Tendo em conta os objetivos determinados no capítulo 1, neste capítulo são descritos os principais módulos da aplicação a desenvolver, bem como o estado da arte relativo aos terminais de bilhética associados ao trabalho do projeto corrente (terminais de bilhética "*fat devices*", e processo de leitura de *smart cards*) e é ainda descrito o processo de escolha do *tablet* com SO *Android*, bem como o do leitor de cartões, necessário para atingir os objetivos propostos.

Um dos principais objetivos do presente trabalho é a separação do processo de leitura de *smart cards* e a lógica de validação. Assim a arquitetura genérica de partida é a seguinte: um leitor de cartões, um dispositivo com SO *Android* e um serviço de validação (ver Figura 5).

De acordo com a arquitetura definida na Figura 5, o desenvolvimento da aplicação proposta é descrito, no capítulo 4, o módulo de leitura de cartões usando uma interface externa ao *tablet* com SO *Android*, no capítulo 5, é descrito o desenvolvimento da camada de apresentação do validador e os processos de interação com a camada externa de leitura e com a camada de validação a qual é descrita no capítulo 6. Sendo assim estes três capítulos são o *core* deste projeto orientado para o desenvolvimento de um validador de títulos de transporte de baixo custo e flexível que permita validar a noção do "*thin device*". Numa área de soluções proprietárias e integradas a proposta do projeto corrente consiste no desenvolvimento de um validador móvel usando um *tablet* com SO *Android*[3]. Este dispositivo permite verificar se um título de transporte é ou não válido.

Este capítulo está organizado nas seguintes secções: 3.1 onde é descrita uma comparação entre os dispositivos existentes atualmente e o novo validador *Android*, implementado no âmbito deste projeto; 3.2 é descrito genericamente o que são comandos e como são utilizados para interagir com o leitor de cartões e os *smart cards* e obter a informação utilizada no processo de validação; 3.3 é fundamentada a escolha

[3] Aplicação desenvolvida para *tablet* com SO *Android* e utilizada para validar títulos de transporte com *smart cards Calypso*.

do leitor de cartões utilizado no âmbito do projeto corrente; 3.4 são apresentadas algumas características relacionadas com os *smart cards Calypso*; 3.5 é fundamentada a escolha do *tablet* com SO *Android* mais adequado para atuar como validador; 3.6 é apresentada a arquitetura final, utilizando os dispositivos escolhidos nos subcapítulos 3.3 e 3.5.

3.1 Validador de Títulos de Transporte – Comparação Entre um "*Fat Device*" e o Validador *Android*

Os validadores existentes atualmente são dispositivos "*fat device*", ou seja, são dispositivos cuja lógica de validação se encontra armazenada localmente no dispositivo e a linguagem de programação utilizada para implementar a lógica de validação é a linguagem C. O preço dos validadores "*fat device*" é elevado (acima dos 1000 euros), sendo que o custo de adaptação da lógica de validação a um novo validador "*fat device*" (proveniente, por exemplo, de outro fornecedor) é também elevado.

Para desenvolver um dispositivo validador "*thin device*" foi necessário adquirir conhecimento sobre o funcionamento deste tipo de dispositivos. Para adquirir o conhecimento necessário foi analisado código escrito na linguagem C que corria num PC (*Personal Computer*) e permitia executar operações sobre *smart cards Calypso* (Lisboa Viva), através de um leitor ligado ao PC por uma porta USB. Apesar de não ser o validador "*fat device*" a ser utilizado em produção, foi possível verificar o comportamento de um validador e analisar também o código já implementado pela empresa Link na linguagem C.

Na Figura 8 são mostradas as camadas de *software* presentes nos dispositivos existentes atualmente ("*fat devices*") e as camadas de *software* presentes no novo validador *Android* implementado no âmbito deste projeto. Pode verificar-se na arquitetura do dispositivo "*fat device*" que a aplicação de validação assenta diretamente sobre a lógica da bilhética, que por sua vez assenta sobre uma camada que permite fazer a comunicação com o leitor. No lado do *tablet* com SO *Android* foi implementada uma biblioteca de interação com leitores e *smart cards*, que incluiu os comandos que podem ser enviados para o leitor e os *smart cards*. Para enviar os comandos foi necessário implementar, para além dos próprios comandos, uma

camada que representa o leitor (*Coupler*) e ainda uma camada usada para enviar e receber os *bytes* relativos aos comandos, através da interface *Bluetooth* (*IOManager*). Estas três camadas de *software* são utilizadas pela aplicação de validação para obter a informação necessária ao processo de validação. Esta informação é posteriormente enviada para um serviço *web* onde é validada e de acordo com a resposta enviada por esse serviço, é mostrado o resultado no ecrã, que indicará se o título de transporte presente no *smart card* é válido.

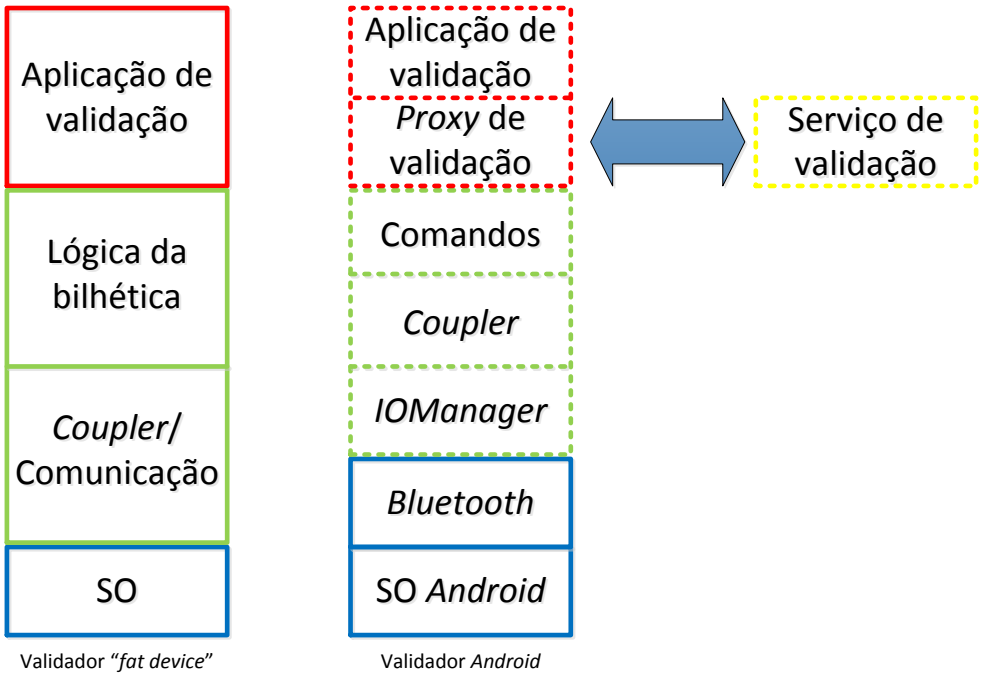

Figura 8 – Camadas de *software* presentes nos "*fat devices*" e no validador *Android*.

No validador "*fat device*" da Figura 8 a camada "Lógica de bilhética" e a camada de "*Coupler*/Comunicação" permitem interagir com leitores e *smart cards*. Do lado do validador *Android* estas funcionalidades são implementadas pelas camadas de *software IOManager*, *Coupler* e pelos comandos. A aplicação de validação assenta sobre as camadas de interação com leitores e *smart cards*, sendo que no validador *Android* a informação obtida através dos comandos é enviada para o serviço de validação onde é validada. Em ambos os dispositivos a camada SO é referente ao SO do respetivo dispositivo e no caso do validador *Android* a camada *Bluetooth* está presente na API do SO *Android*. As camadas de *software* a tracejado foram implementadas no contexto do projeto corrente.

3.2 Validar um Título de Transporte Presente num *Smart Card*

Para validar um título de transporte eletrónico, é necessário obter informação presente no *smart card* do respetivo título, que é posteriormente enviada para o serviço de validação. A informação utilizada no processo de validação está armazenada sob a forma de contratos que estão presentes nos *smart cards*. Para validar um título de transporte é então necessário solicitar um ou mais contratos ao respetivo *smart card* e verificar posteriormente a validade de cada um dos contratos.

O processo de leitura dos contratos, presentes no *smart card*, é feito através de um intermediário, designado por leitor de cartões, o qual reencaminha as instruções recebidos do validador *Android* para o respetivo *smart card*. As instruções enviadas para o leitor e para os *smart cards*, são designadas de comandos.

Neste subcapítulo é também abordado genericamente o que são comandos e como podem ser utilizados para obter um ou mais contratos presentes nos *smarts cards Calypso*.

3.2.1 Processo de Leitura de um Contrato

A comunicação com o leitor de cartões é realizada através de instruções, denominadas de comandos, às quais este envia uma resposta.

Cada leitor tem comandos específicos, que podem por exemplo ser utilizados, para ligar ou desligar a antena do leitor ou obter um identificador do *firmware* utilizado pelo leitor em causa.

A comunicação com os *smart cards* também é realizada através de comandos, que são específicos de *smart card* para *smart card* e que são enviados utilizando o leitor de cartões como intermediário.

O processo de leitura consiste em enviar comandos, para o leitor e para o *smart card* do título de transporte em causa, sendo que na resposta aos comandos vem a informação necessária para o processo de validação, nomeadamente os contratos presentes no *smart card* e o número de série do mesmo.

3.2.2 Tipos de Comandos

Existem dois tipos de comandos, os comandos enviados para o leitor e os comandos enviados para o *smart card.* Um comando, enviado para o leitor, é constituído por uma classe (CLA), pela instrução (INS) do comando, e pelos parâmetros (DATA) do comando tal como demonstra a Figura 9. Cada um destes campos é representado por um *byte* que é envidado para o leitor, sendo que a classe e a instrução são utilizadas para identificar a operação que se pretende utilizar [11].

Figura 9 – Formato de um comando.

Para enviar um comando é necessário encapsular os *bytes* já referidos entre um ou mais *bytes* de *header* (SOF (*Start of Frame*)) e um ou mais *bytes* de *trailer* (EOF (*End of Frame*)). No final é necessário calcular um código CRC com um algoritmo específico do próprio leitor e enviar também esse código CRC. Este código é utilizado pelo leitor para saber se ocorreu algum erro na transmissão do comando do *tablet* com SO *Android* para o leitor.

As respostas têm o mesmo formato que os comandos, sendo que nos dados da resposta é indicado, por exemplo, se o comando foi realizado com sucesso. Caso o comando solicite informação, a informação solicitada é enviada pelo *smart card* nos dados da resposta.

3.2.2.1 APDU (*Application Protocol Data Unit*)

Os comados que são enviados diretamente para o *smart card* são designados de APDU (*Application Protocol Data Unit*) e têm um formato próprio, que inclui uma classe e uma instrução próprias [12].

Um dos comandos presente em alguns leitores permite enviar (“redirecionar”) um comando APDU diretamente para o *smart card* e receber por sua vez a resposta gerada pelo mesmo *smart card*. Desta forma é possível implementar comandos para interagir com o mesmo tipo de *smart card* utilizando leitores de fornecedores

diferentes. Ou seja, o comando que é enviado para o *smart card* depende apenas deste e não do leitor que intermedia a comunicação com o mesmo.

Apesar dos comandos APDU estarem desenhados para ser enviados para o *smart card*, necessitam de ser enviados através de um leitor cartões apropriado, sendo assim, o leitor é o intermediário responsável por encaminhar esses comandos para o *smart card* em causa. O comando enviado para o leitor é construído de forma implícita pela biblioteca de interação com leitores e *smart cards* sempre que se pretende enviar um comando APDU. Assim um comando APDU tem o formato representado pela Figura 10, sendo os *bytes* a exteriores refentes ao comando enviado para o leitor e os *bytes* no retângulo, com a linha a negrito, são os *bytes* referentes ao comando APDU que são reencaminhados pelo leitor para o *smart card*.

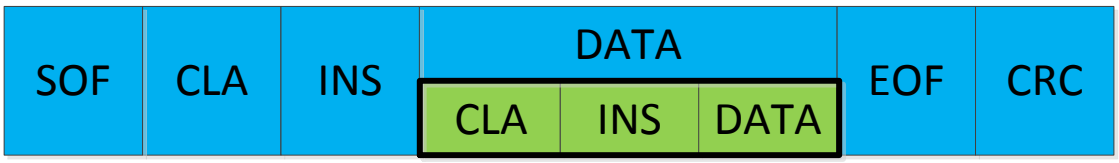

Figura 10 – Formato de uma mensagem APDU.

3.3 Processo de Escolha do Leitor de Cartões

Nesta secção é descrito o processo de escolha do leitor de cartões (também designado de *coupler*) a ser usado no presente trabalho. Tendo em conta a experiência da empresa Link, a primeira solução a ser testada foi um leitor ASK, que já era utilizado anteriormente pela empresa Link. Este leitor é utilizado na OTLIS (Operadores de Transportes da Região de Lisboa) e na STIB (*Société des Transports Intercommunaux de Bruxelles*) para o processo de personalização de *smart cards*. Desta forma o conjunto de comandos que este leitor suporta eram bem conhecidos da empresa Link, o que permitiu uma aprendizagem mais rápida da minha parte. Este processo demorou cerca de três semanas.

No entanto, apesar do leitor funcionar corretamente (quando ligado a um computador), a comunicação com este leitor era realizada através de uma porta USB (*Universal Serial Bus*).

A fase seguinte do teste foi a interação com o *tablet* e aí ocorreu um problema, porque o *tablet* com SO *Android* não permite que seja acedido ao leitor via USB (sem que o utilizador tenha privilégios de administrador no *tablet* com SO *Android*).

Face a esta contrariedade, foi necessário iniciar o processo de escolha de outro leitor de cartões, com o qual fosse possível evitar a comunicação via USB. Uma alternativa a este processo de comunicação seria através de *Bluetooth*. Este processo apresenta a vantagem de não existir ligação física, no entanto a velocidade de transmissão de dados poderá ser inferior.

Da análise feita ao longo de três dias, na qual foram analisadas as especificações de vários equipamentos a escolha recaiu sobre o leitor AEP [13], o qual tem um conjunto de instruções idêntico ao leitor ASK e comunica com o *tablet* com SO *Android* através de *Bluetooth*. Os testes iniciais permitiram ter a certeza que esta seria uma boa solução, uma vez que permitiram confirmar o bom comportamento do leitor na interação com os *smart cards Calypso*, pelo que a empresa Link procedeu a aquisição do *hardware*, que ficou disponível para ser utilizado no projeto corrente a partir de Fevereiro de 2013.

3.4 Caraterísticas do *Smart Card Calypso*

Neste subcapítulo são abordadas algumas características presentes nos *smart cards Calypso*, o qual constituiu o caso de teste, dado os interesses comerciais da empresa Link. Inicialmente são abordadas as interfaces que os *smart cards Calypso* têm para comunicar com os leitores existentes. De seguida são descritos os tipos de estruturas de dados presentes no "sistema de ficheiros" do *smart card Calypso*. No último subcapítulo são abordados genericamente os contratos e como podem ser obtidos a partir de um *smart card Calypso* [11].

3.4.1 Modos de Comunicação Suportadas pelo *Smart Card Calypso*

Cada *smart card Calypso* está equipado com uma antena e com uma interface de contacto, permitindo desta forma que o *smart card* possa interagir tanto com leitores de contacto, como com leitores sem contacto.

A tecnologia sem contacto utiliza ondas na frequência dos 13.56 MHz e permite transferência de dados na ordem dos 848 Kbps e implementa a norma ISO 14 443 [10].

O leitor AEP utilizado no contexto deste projeto suporta apenas a interface sem contacto.

3.4.2 Comunicação Segura com *Smart Card Calypso*

Uma particularidade dos *smart cards Calypso* consiste no facto de se poder ler a informação, presente no *smart card,* sem qualquer tipo de autenticação por parte do leitor.

Isto significa que não se pode confiar no conteúdo que o *smart card* envia (sem autenticação), ou seja, um *smart card* pode enviar informação que foi "copiada" de um *smart card* válido, permitindo eventualmente desta forma que existissem títulos de transporte forjados.

No projeto corrente não foi utilizado o módulo de segurança SAM, pelo que a leitura realizada do *smart card* não é feita de forma segura, no entanto, num projeto real não se deve utilizar a leitura de informação de forma desprotegida, mas sim a leitura com autenticação. Para garantir a autenticação é necessário utilizar um modo de leitura, onde o *smart card* precisa de enviar a informação, juntamente com uma assinatura, que garante que a informação enviada é autêntica. Ou seja, a leitura deve ser realizada no contexto de uma sessão, desta forma quando a sessão é fechada é verificada a autenticação do *smart card*.

3.4.3 Estrutura de Dados num *Smart Card Calypso*

É possível guardar informação dentro de um *smart card Calypso*, como por exemplo os contratos atualmente ativos.

A estrutura do sistema de ficheiros está organizada em dois tipos de ficheiros: os *dedicated files* (ficheiros dedicados) e os *elementary files* (ficheiros elementares). Um ficheiro dedicado pode também ser chamado de "*directory*" *file*, e é análogo a uma pasta sendo que no seu conteúdo podem tanto estar ficheiros elementares como outros ficheiros dedicados [11].

Um ficheiro elementar consiste num ficheiro cujo conteúdo são dados, e pode ser um de quatro tipos: linear, cíclico, binário e contador.

Existem duas pastas pré-definidas: a pasta de raiz denominada de *master file* e uma pasta especial que contém informação relativa à *application Calypso*, denominada de *Calypso dedicated file* (diretório *Calypso*).

No diretório *Calypso* estão guardadas informações que podem ser utilizadas pela operadora de transportes para guardar, entre outras coisas, os contratos ou um registo das atividades recentes.

Para verificar se um título de transporte é válido é necessário então consultar todos os contratos guardados no ficheiro denominado de "*Contracts*" que está dentro do diretório *Calypso* presente no *smart card* e verificar se algum deles é válido para o operador corrente.

3.4.4 Tipos de Ficheiros Elementares

Como foi dito anteriormente existem quatro tipos de ficheiros elementares: linear, cíclico, binário e contador [11].

A unidade mínima de atribuição no "sistema de ficheiros" de um *smart card Calypso* é de um *record* (para os ficheiros lineares, cíclicos e contadores), sendo que cada *record* pode armazenar até 255 *bytes*.

Os ficheiros do tipo contador são ficheiros que guardam apenas um inteiro que é incrementado ou decrementado. O número de contadores diferentes no mesmo ficheiro é o resultado do número de *records* total desse ficheiro a dividir por três. Um exemplo da utilização de um ficheiro elementar do tipo contador é o bilhete 7 Colinas, em que apenas é decrementada uma unidade em cada viagem, e incrementado o número de unidades quando o título é recarregado.

Os ficheiros lineares contêm um número de fixo de *records* que podem ser lidos e escritos livremente. Um exemplo de um ficheiro linear é o ficheiro "*Contracts*", já referido anteriormente e que está presente no diretório *Calypso* do *smart card*.

O terceiro tipo de ficheiro que utiliza *records* para guardar informação é o ficheiro cíclico. Um ficheiro cíclico é um ficheiro que funciona como uma fila FIFO (*First In First Out*) de *records* com um tamanho fixo. Nestes tipos de ficheiros, um novo *record* é sempre adicionado à cabeça da lista, sendo que o *record* que estava anteriormente na cabeça da lista passa a ficar na segunda posição da lista, o record que estava na segunda posição passa a ficar na terceira posição, e assim sucessivamente até que o último *record* é descartado. Este tipo de ficheiros pode ser utilizado, por exemplo, para guardar um registo da atividade recente do *smart card*.

Por fim existem os ficheiros binários, que permitem guardar um conjunto de *bytes* que pode ser acedido e modificado livremente.

3.4.5 *Applications*

As estruturas de dados (ficheiros e diretórios) presentes num *smart card Calypso* estão definidas dentro uma ou mais *applications* (aplicações) [11].

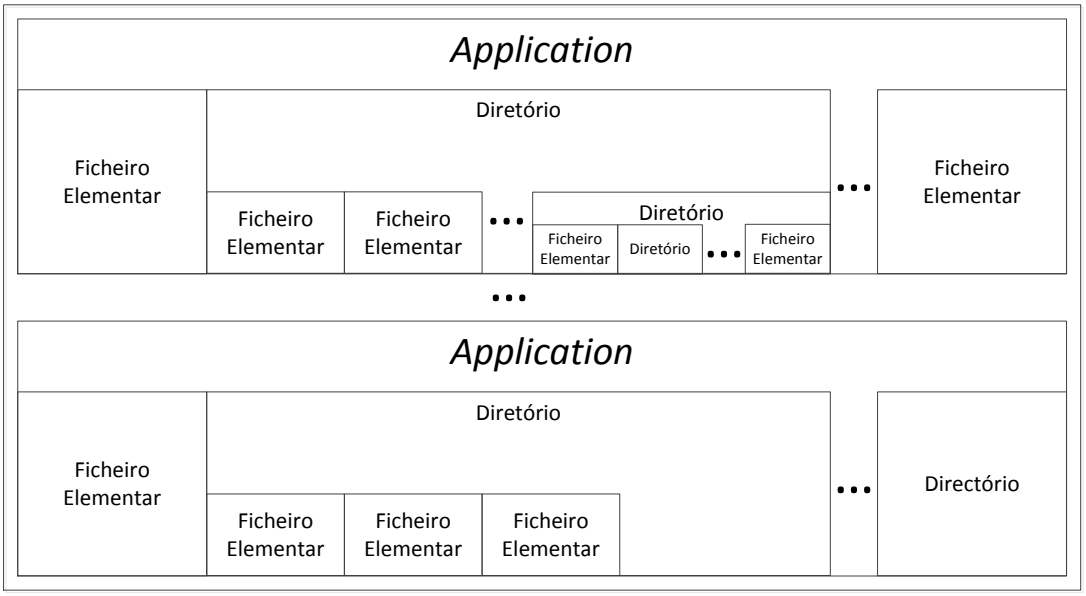

Figura 11 – Sistema de ficheiros disponível no *smart card Calypso*.

Na Figura 11 pode observar-se um exemplo de uma *application*, que guarda alguns ficheiros e diretórios. Pode-se também observar que um diretório (ficheiro dedicado), também pode conter ficheiros elementares ou ainda outros diretórios.

Desta forma para aceder aos ficheiros presentes numa *application* é necessário selecionar essa mesma *application*, e só depois é possível aceder aos dados neles

contidos. Para selecionar uma *application* é necessário enviar o comando *select application* para o cartão indicando o *application identifier* (identificador de aplicação) a que se pretende aceder.

A resposta ao comando *select application* contém a estrutura da *application* selecionada, juntamente com alguma informação que indica por exemplo o número de série do *smart card*.

Um *application identifier* é único e consiste num *array* de *bytes* com tamanho variável com um mínimo de oito *bytes* e um máximo de dezasseis *bytes*. Para selecionar uma *application* pode ser envidado para o cartão tanto o identificador completo, ou podem ser enviados apenas os primeiros oito *bytes*, sendo que neste caso é selecionada a *application* que contém os primeiros oito *bytes* iguais ao identificador enviado no comando.

Para definir uma nova *application* é recomendado que seja utilizado um identificador único, devendo por isso o mesmo ser registado. O registo tem como objetivo garantir que o identificador é único, prevenindo assim que exista um conflito entre duas operadoras diferentes que "acidentalmente" definem o mesmo identificador para a sua *application*, gerando assim um conflito se ambas as aplicações estivem a ser alojadas no mesmo *smart card*.

3.4.6 Aceder a Ficheiros

Após selecionar uma *application* é possível ler e modificar o conteúdo dos ficheiros presentes na *application* selecionada. É portanto necessário identificar o ficheiro a que se pretende aceder. Existem duas formas de identificar um ficheiro: a primeira consiste em utilizar o um *long file identifier*, que é constituído por dois *bytes*, que são únicos dentro do diretório onde o ficheiro de se encontra. Se for necessário, é possível indicar o caminho completo para um ficheiro, indicando um *array* com todos os identificadores dos diretórios do caminho desde a raiz até ao ficheiro pretendido.

A segunda forma de identificar um ficheiro é utilizar um SFI (*Short File Identifier)*, que consiste num valor entre zero e trinta que identifica um ficheiro dentro de um

diretório. Este tipo de identificador só pode ser utilizado em ficheiros elementares descritos na secção 3.4.3 e na secção 3.4.4.

Não é necessário enviar em todos comandos o identificador do ficheiro (elementar ou diretório), ou seja, é possível em algumas situações omitir o identificador da *application* e do ficheiro, sendo que as operações são realizadas sobre o ficheiro atualmente marcado como ficheiro corrente. O ficheiro corrente pode ser, por exemplo, o último ficheiro elementar selecionado ou utilizado, sendo o diretório onde o ficheiro se encontra considerado o diretório corrente e a *application* a que o diretório pertence é considerada a *application* corrente.

3.4.6.1 Modos de Acesso a Ficheiros

Cada ficheiro pode conter um de quatro modos de acesso distintos (*always*, *never*, *session* e PIN), que permitem controlar o acesso a esse ficheiro. Desta forma um ficheiro contém uma estrutura que indica qual o modo de acesso para um dado comando, poder aceder ou alterar a informação presente no respetivo ficheiro.

O primeiro modo de acesso é o *always*, que quando aplicado a um comando, garante que esse comando pode ser sempre executado sobre o ficheiro em causa.

Por sua vez o modo *never* garante que um comando nunca pode ser executado sobre um dado ficheiro.

O modo *session* indica que o comando só pode ser executado dentro de uma sessão segura criada entre o terminal e o *smart card*. Neste tipo de modo de acesso é necessário indicar qual a chave menos privilegiada com autorização para executar o comando durante uma sessão.

O quarto modo é opcional e consiste em utilizar um código PIN que garante que um comando só pode ser executado após a introdução desse código PIN que é validado pelo *smart card*.

3.4.7 Contratos

Um contrato é um conjunto de informações relativas a um título de transporte, que pode ser válido em um ou mais operadores de transportes.

Cada contrato pode conter uma data de validade a partir da qual é considerado inválido, sendo necessário recarregar o título para que seja utilizado novamente.

A *application Calypso* suporta até quatro contratos diferentes no ficheiro denominado de "*Contracts*".

3.5 Processo de Escolha do *Tablet* com SO *Android*

Uma das primeiras tarefas relacionadas com a realização do projeto corrente consistiu na escolha do *tablet* com SO *Android* mais adequado para ser utilizado como validador.

Foram analisadas várias características em nove *tablets* diferentes. As principais características analisadas consistiram no número de portas USB (*Universal Serial Bus*) existentes no dispositivo, possibilidade de ser um dispositivo USB *host*, formas de comunicação sem fios presentes no dispositivo, tamanho do ecrã e versão do sistema operativo *Android*.

O facto do *tablet* com SO *Android* deter duas portas USB era relevante uma vez que inicialmente se pretendia ligar um leitor através de uma porta USB. Deveria ainda ser possível executar a aplicação, implementada no contexto deste projeto, em *debug* no *tablet Android*. Como para correr a aplicação em *debug* é necessário que o computador esteja ligado ao *tablet* através de uma porta USB, o *tablet* necessitaria de disponibilizar no mínimo duas portas USB.

Outra característica relevante relativa ao USB seria o facto de permitir que o *tablet* com SO *Android* funcionasse como um *host*, permitindo desta forma que fosse possível conectar o leitor de cartões ao *tablet*. Como já foi dito anteriormente, apesar de alguns dos dispositivos permitirem *host* de determinados periféricos como ratos ou teclados, não foi possível ligar o leitor via USB ao *tablet Android* uma vez que não se evidenciou, que o leitor seria reconhecido.

No entanto, como já foi referido anteriormente, o *tablet Android* escolhido não comunica com o leitor através de USB, mas sim através de Bluetooth.

Para validar um título de transporte é necessário que o *tablet* com SO *Android*, comunique com um serviço de validação, sendo assim as duas formas de comunicação por Wi-Fi e/ou 3G são relevantes no dispositivo.

Das várias cateterísticas que um *tablet Android* deveria ter, a que poderia ser mais útil era a capacidade de utilizar o NFC (*Near Field Communication*). O NFC permite a interação com dispositivos que utilizam a tecnologia ISO 14443, que é o caso dos *smart cards* utilizados no âmbito deste projeto.

Na Tabela 1 podem observar-se os dispositivos analisados no âmbito do projeto corrente, juntamente com as características consideradas relevantes que cada dispositivo tem. As células marcadas com traço (-) indicam que não foi possível confirmar a presença, ou não, da funcionalidade em causa, através da documentação disponibilizada pelo fabricante.

Tabela 1 – Dispositivos móveis analisados.

Dispositivo	***USB Host***	***Wi-Fi***	***Bluetooth***	**3G**	**Tamanho do ecrã**	**Sistema Operativo (*Android*)**	**Suporte ao desenvolvimento (SDK)**	**NFC**	**Alimentação dedicada**
Sony Tablet S (SGPT113PT/S)	Sim	Sim	Sim	Sim	9.4”	3.1 (expansível até 4.0.3)	-	Não	Sim
Xperia Tablet S (SGPT131E1)	Sim	Sim	Sim	Sim	9.4”	4.0.3	-	Não	Não
TOSHIBA AT300-101	Sim	Sim	Sim	Não	10.1”	4.0	Sim	Não	Sim
Galaxy Tab 2	-	Sim	Sim	Não	10.1”	4.0	Sim	Não	Não
ARCHOS 101 G9	Sim	Sim	Sim	Sim	10.1”	4.0	Sim	Não	Tem duas portas USB
ARCHOS 80 G9	Sim	Sim	Sim	Sim	8”	4.0	Sim	Não	Tem duas portas USB
Pidion BP70	-	Sim	Sim	Sim	10.1”	4.0	-	Sim	-
Pidion BP50	-	Sim	Sim	Sim	7”	4.0	-	Sim	-
Nexus 7	Sim	Sim	Sim	Sim	7”	4.1	Sim	Sim	Não

A escolha do *tablet Android* mais adequado recaiu então sobre o *Asus Google Nexus 7* pois era o dispositivo que apresentava características mais adequadas como o NFC, o

Wi-Fi, o 3G e a versão mais recente do SO *Android*. No entanto pelos motivos que são indicados no capítulo 4, a tecnologia NFC não foi a tecnologia utilizada para comunicar com os *smart cards*.

3.6 Arquitetura da Solução Implementada

O *tablet* com SO *Android* é um *tablet Asus Google Nexus 7*, equipado com um leitor de cartões AEP. Este *tablet* com SO *Android* permite validar os contratos presentes em *smart cards Calypso* (utilizados no cartão Lisboa Viva), mostrando o resultado da validação no ecrã do *tablet*.

O *tablet* com OS *Android* comunica com *smart cards* utilizando um leitor AEP, com o qual interage através *Bluetooth*. Por sua vez o leitor comunica com os *smart cards* utilizando a norma para comunicação sem contacto ISO 14443 (*Innovatron Radio Protocol*).

A aplicação de validação desenvolvida para o *tablet* com SO *Android* é dirigida a revisores e permite que os mesmos verifiquem a validade de um título de transporte de um passageiro (ver Figura 8 e Figura 12).

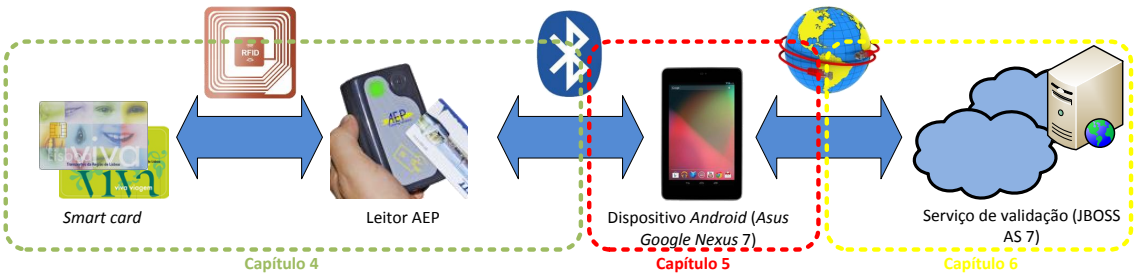

Figura 12 – Arquitetura da solução (dividida por capítulos).

Tal como demonstra a Figura 12 inicialmente o passageiro aproxima o título de transporte ao leitor. O validador *Android* é então responsável por obter informação do título de transporte e envia-la para o serviço de validação utilizando a *Internet*. Consoante a resposta do serviço é mostrado no ecrã do validador *Android* se o título de transporte é ou não válido.

4 Interface Externa de Leitura de *Smart Cards*

Neste capítulo é apresentado o trabalho desenvolvido relacionado com o processo de leitura dos *smart cards Calypso* (Lisboa Viva).

Como já foi dito anteriormente, no contexto deste projeto é necessário comunicar com um leitor e *smart cards* para ler a informação presente nos mesmos e validar posteriormente se estes são ou não títulos de transportes válidos. Desta forma, foram realizadas várias abordagens a fim de implementar uma biblioteca em *Java* de interação com o leitor e os *smart cards*. Assim nesta secção é descrita a biblioteca de interação com o leitor e os *smart cards* implementada para o *tablet* com SO *Android* após a análise dos terminais existentes. Esta biblioteca desempenha algumas funcionalidades da biblioteca em C da empresa Link, define uma arquitetura orientada por objetos e permite que seja mais fácil a inclusão de novos leitores e *smart cards*.

Tal como demonstra a Figura 13, esta biblioteca é utilizada pelo *tablet* com SO *Android* para poder obter as informações necessárias ao processo de validação de um título de transporte.

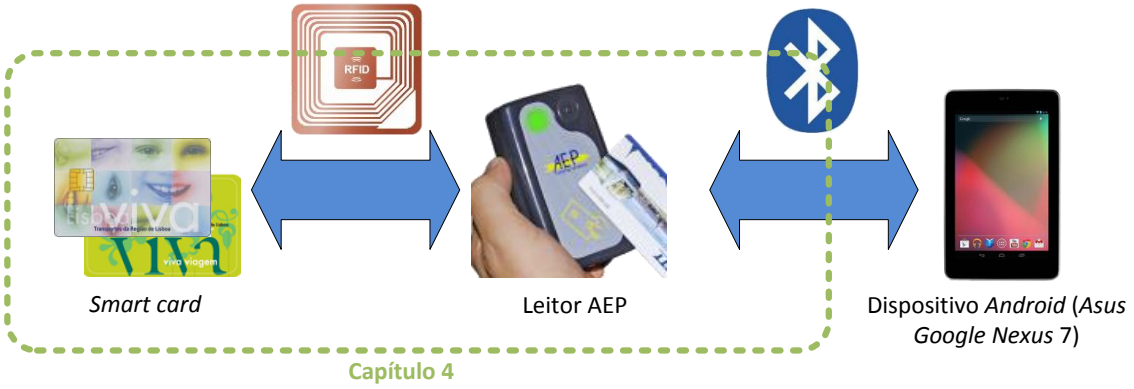

Figura 13 – Arquitetura da camada externa.

Este desenvolvimento beneficiou muito da minha presença na empresa Link durante o período de Janeiro de 2012 a Junho de 2013 e da assistência dada pelo engenheiro Hugo Bicho (da empresa Link).

Salienta-se que a biblioteca de interação com leitores e *smart cards* desenvolvida no âmbito deste projeto, encontra-se atualmente a ser utilizada no projeto da Link

SmartCITIES *Cloud Ticketing*, realizando operações em ambiente de produção como o carregamento de cartões Lisboa Viva.

Este capítulo está dividido em 8 secções: 4.1 e 4.2 onde são descritas as várias abordagens utilizadas com o objetivo de conseguir desenvolver a biblioteca de interação com leitores e *smart cards*; 4.3 onde é indicada qual a consequência de não poder ser utilizado o leitor NFC presente no *tablet* com SO *Android*; 4.4 a 4.6 são descritas os componentes existentes na biblioteca implementada, que consiste: (1) nos comandos, (2) na interface de comunicação de mais baixo nível e o (3) mecanismo utilizado para interpretar as respostas aos comandos; 4.7 são descritos os vários comandos implementados no âmbito do projeto corrente e que são utilizados pelo validador *Android*; 4.8 é descrito um problema existente na API *Bluetooth* do *Android* e como foi ultrapassado.

4.1 Abordagens Iniciais

A primeira abordagem consistiu em realizar o código de interação com o leitor na linguagem C, e posteriormente utilizar o JNI (*Java Native Interface*) para ligar a lógica em *Android* com o código de acesso a leitores e *smart cards*. Para compilar o código utilizou-se o *Android* NDK (*Native Development Kit*).

Nesta abordagem seria mais simples integrar o código desenvolvido para *Android*[4] com o código já implementado pela empresa Link, uma vez que o código utilizado pela Link para interação com leitores e *smart cards* está também implementado também em C.

Esta abordagem não foi bem-sucedida, uma vez que o código que permite interagir com o leitor necessita de utilizar uma porta COM e não foi possível aceder à porta COM nos testes realizados. Isto aconteceu porque para aceder à porta COM é necessário ter privilégios de administrador no *tablet* com SO *Android*.

A segunda abordagem consistia em implementar código *Java*, mas que utilizaria a biblioteca NFC disponibilizada pelo próprio SO *Android*. Esta abordagem também não foi bem-sucedida, pois o leitor de NFC integrado no *Asus Google Nexus* 7 não fornece energia suficiente para o bom funcionamento dos *smart cards Calypso*.

[4] Código da aplicação desenvolvida para o *tablet* com SO *Android*.

4.2 Abordagem Final

Face aos problemas identificados na secção anterior, optou-se pela aproximação usando uma biblioteca em *Java* que permite que sejam enviados comandos para o leitor e que sejam recebidas e interpretadas as respetivas respostas. Esta biblioteca permite abstrair o programador da lógica de interação com o leitor e os *smart cards*, e é representada pelo diagrama de classes ilustrado pela Figura 14.

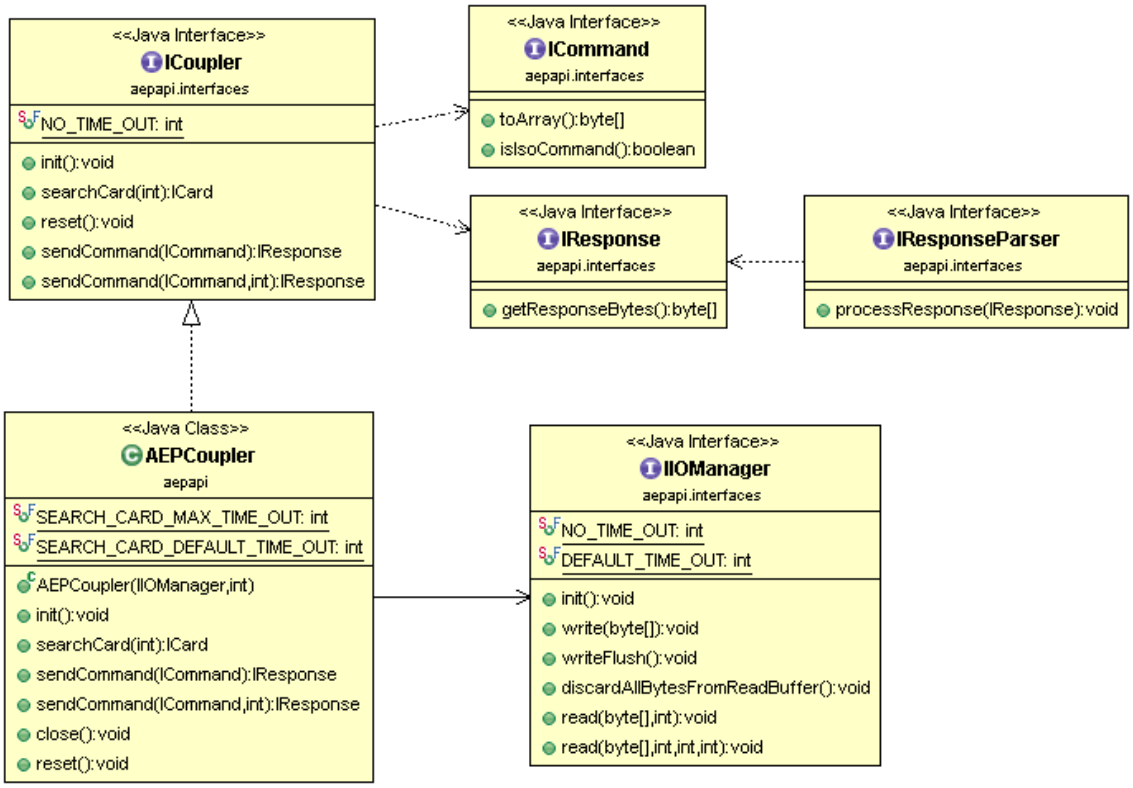

Figura 14 – Diagrama de classes da biblioteca implementada.

Esta biblioteca permite que sejam criados comandos, e que os mesmos sejam enviados para o leitor, sendo que, de seguida é interpretada a resposta proveniente do leitor ao comando enviado.

A biblioteca pode ser adaptada facilmente a vários tipos de leitores e a diferentes formas de comunicação.

Dentro da aplicação *Android*[5] esta biblioteca utiliza uma porta COM disponibilizada pelo próprio SO *Android* para comunicar com o leitor *Bluetooth*.

[5] Aplicação de validação desenvolvida para o *tablet* com SO *Android*.

Enviar um comando consiste em enviar (escrever) os *bytes* desse comando (juntamente com os *bytes* já referidos no formato previamente referido) para a porta COM do dispositivo *Bluetooth*, sendo que as respostas aos comandos são recebidas (lidas) na mesma porta COM.

É suportado ainda o envio de um comando mas com um valor de *timeout* que permite que a aplicação não fique bloqueada indefinidamente à espera da resposta enviada pelo leitor.

4.3 Enquadramento da Biblioteca Implementada na Aplicação de Validação

A utilização de NFC facilitaria o desenho e utilização de comandos para a interação com *smart cards,* uma vez que tem suporte nativo por parte do SO *Android* e o leitor NFC já se encontra incorporado no *tablet* com SO *Android* utilizado no âmbito deste projeto.

No entanto não foi possível utilizar NFC, pelo motivo já referido anteriormente. Sendo por isso então necessário implementar uma biblioteca que permitisse interagir com o leitor e os *smart cards Calypso*. Esta biblioteca *Java* permite que sejam enviados comandos e é utilizada pela aplicação para obter a informação necessária para o processo de validação, que é posteriormente enviada para o serviço de validação.

Esta biblioteca *Java*, identificada na Figura 15 pelo retângulo a negrito, assenta sobre as bibliotecas de *Bluetooth* disponibilizadas pelo SO *Android*, e ao contrário do cenário onde seria utilizado o NFC, é necessário realizar código mais próximo do *hardware* para interagir com o leitor e os *smart cards*. Este código inclui encapsular os bytes relativos aos comandos com os *bytes* de *header* e *trailer*. É necessário também controlar o leitor realizando operações como pedir ao leitor para ligar a antena e procurar um cartão, ou controlar o mecanismo de poupança de energia do leitor.

Esta biblioteca suporta também funcionalidades como calcular o CRC para os comandos enviados e verificação do CRC para as respostas aos respetivos comandos. É possível também interpretar as respostas aos comandos, convertendo-as para um

formato mais fácil de usar pelo programador, informando-o se necessário da ocorrência de um erro na execução do comando.

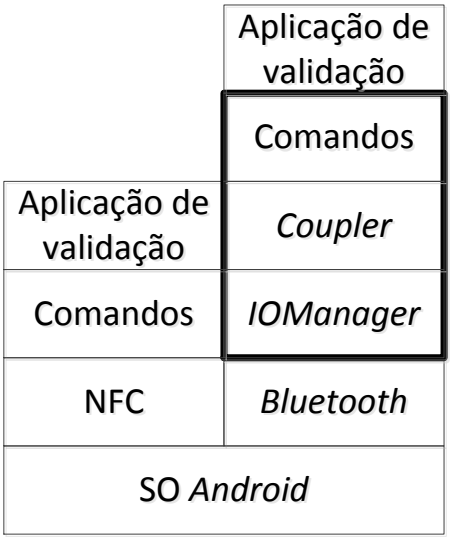

Figura 15 – Comparação entre a biblioteca implementada e a biblioteca NFC.

4.4 Comandos de Interação com o Leitor e *Smart Cards*

Como foi dito anteriormente as instruções são enviadas para o leitor e os *smart cards*, sobre a forma de comandos. Um comando consiste numa classe cujo código é capaz converter uma instrução que o programador pretende enviar para o leitor ou para o *smart card*. Para isso é necessário converter a instrução que o programador pretende enviar para um *array* de *bytes* que é posteriormente enviado para o leitor. Todos os comandos implementam a interface *ICommand* (ver Figura 16 e Figura 17).

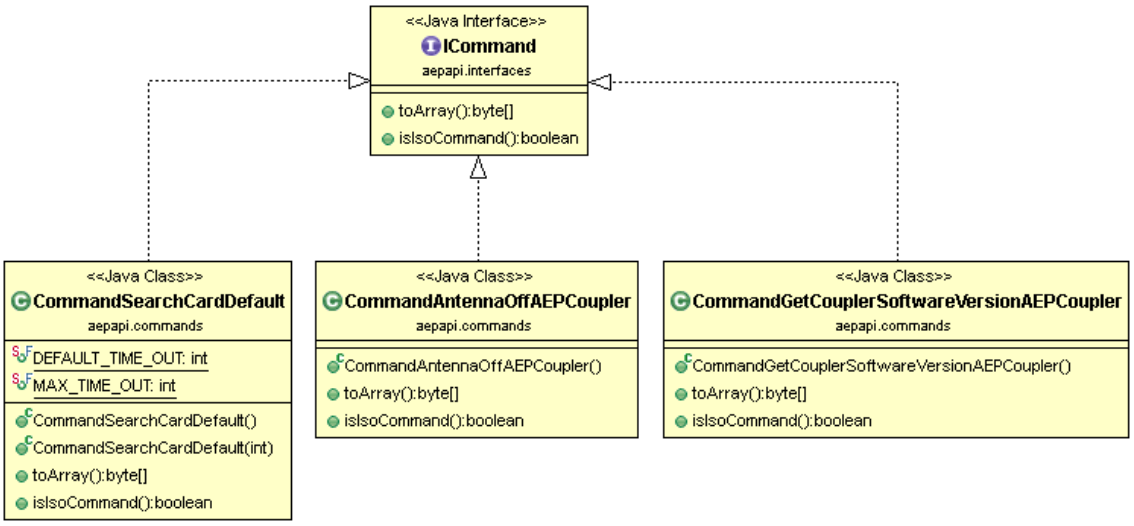

Figura 16 – Comandos *search card*, *antenna off* e *get software version*.

Um comando é um objeto subtipo de *ICommand*, tendo por isso que implementar dois métodos relevantes: o método *toArray*, e o método *isIsoCommand*.

O método *toArray* retorna o *array* de bytes que representa o comando e que é enviado para o leitor. Os comandos que recebem parâmetros, como por exemplo o identificador da *application* que se pretende selecionar, constroem o *array* de *bytes* de acordo com os parâmetros recebidos.

O segundo método retorna um booleano e é utilizado para indicar se um comando é um *IsoCommand*. O conceito de *IsoCommand* consiste num comando APDU, ou seja, os *bytes* que constituem o comando corrente deverão ser enviados para o cartão. Os *bytes* (APDU) gerados por um *IsoCommand* são independentes do leitor utilizado, o que permite que funcionem através de qualquer leitor. Com este tipo de comandos é possível implementar um comando tendo apenas como preocupação os *bytes* que são enviados para o *smart card*, e nas respostas os *bytes* recebidos diretamente do cartão. Para este tipo de comandos a biblioteca "encarrega-se" de acrescentar e formatar os *bytes* necessários para o envio deste comando.

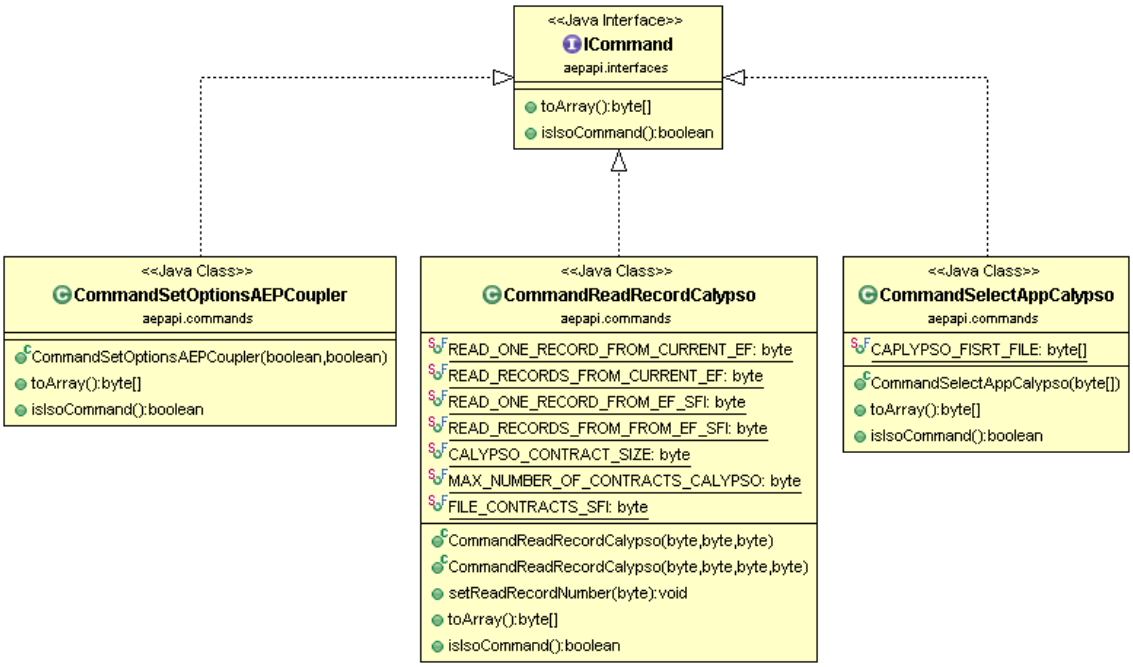

Figura 17 – Comandos *set options*, *read record* e *select application*.

4.5 Interface de Comunicação com o Leitor

Um objeto do tipo *ICoupler* representa um leitor e consiste numa classe que contém métodos públicos que permitem realizar alguns comandos relevantes, como por exemplo executar o comando que permite procurar um cartão ou o comando que permite realizar o *reset* ao leitor.

Na implementação para o leitor AEP, o método que realiza a operação de *reset* não realiza qualquer operação, uma vez que com este leitor não é necessário realizar qualquer operação de *reset*.

Foi implementado ainda um método que permite enviar um comando e retorna a resposta a esse comando. Neste método são obtidos os *bytes* do comando que se pretende executar, sendo posteriormente adicionada informação, nomeadamente os *bytes* que são específicos do leitor como os *bytes* de *header* e *trailer.* É também calculado para cada comando um código CRC de dois *bytes* com vista a garantir que o comando chega sem erros ao leitor.

Este método é também responsável por formatar os comandos que são do tipo *IsoCommand* de maneira a que o leitor consiga enviar estes comandos para o *smart card*. Para isso é necessário transformar o comando que contém os *bytes* do APDU num comando que possa ser interpretado pelo leitor, permitindo desta forma ao leitor "reencaminhar" o comando APDU para o *smart card*.

Para enviar um comando é utilizada uma classe de mais baixo nível denominada de *IOManager*, que é responsável apenas por enviar e receber *bytes*.

Por cada comando que é enviado é recebida uma resposta. É neste método também que é lida a resposta ao comando previamente enviado. Inicialmente é lido o *status code* que indica se ocorreu, ou não, um erro quando o comando foi executado. De seguida é lido o número de *bytes* que indica o tamanho da resposta, e são então lidos todos os *bytes* da resposta.

Após todos os *bytes* terem sido lidos, são então lidos dois *bytes* finais que indicam o CRC da resposta e é então feita uma primeira análise à integridade da mesma que

consiste em calcular o CRC da resposta e verificar se não ocorreram erros na transmissão.

Se após a verificação não tiverem ocorrido erros, são então removidos os *bytes* extra presentes na resposta, ou seja, é removido o CRC, os *bytes* de *header* e os *bytes* de trailer. Caso o comando tenha sido um APDU (*IsoCommand*) a resposta passa conter apenas os *bytes* que fazem efetivamente parte da resposta, ou seja, os *bytes* que foram enviados pelo cartão. No final, são retornados os *bytes* da resposta.

4.6 Interpretar Respostas aos Comandos

Como foi dito anteriormente para executar um comando é necessário criar um objeto do tipo comando, que posteriormente é enviado para o leitor. Desta forma é necessário interpretar a resposta ao comando, para que o resultado seja convertido num formato mais simples de entender e usar pelo programador.

Uma classe que permite interpretar uma resposta tem o nome de *response parser*, e para cada comando existe uma classe que interpreta a resposta a esse mesmo comando. Na Figura 18 e na Figura 19 pode observar-se que todos os *response parsers* implementam a interface *IResponseParser*.

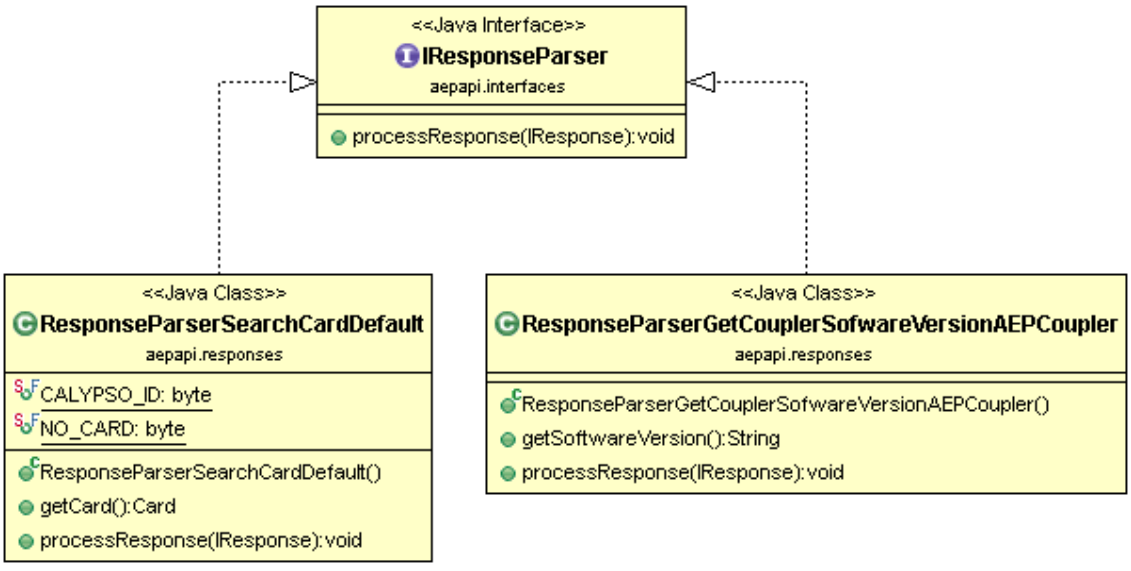

Figura 18 – *Response parsers* para os comandos *search card* e *get software version*.

Para interpretar uma resposta é inicialmente verificado se a resposta se encontra num formato válido e em caso afirmativo é iniciado o processo de interpretação da resposta.

Uma resposta pode ser apresentada em vários formatos. Um dos formatos utilizados corresponde a um TLD (*Type Length Data*), sendo que o primeiro campo (*type*) é usado para indicar o tipo da resposta, o segundo campo (*length*) indica o tamanho da resposta e o terceiro campo (*data*) contém uma parte do conteúdo da resposta.

Desta forma uma resposta pode ser constituída por um ou mais campos do tipo TLD sendo que o campo *data* pode ter por sua vez o conteúdo também no formato TLD.

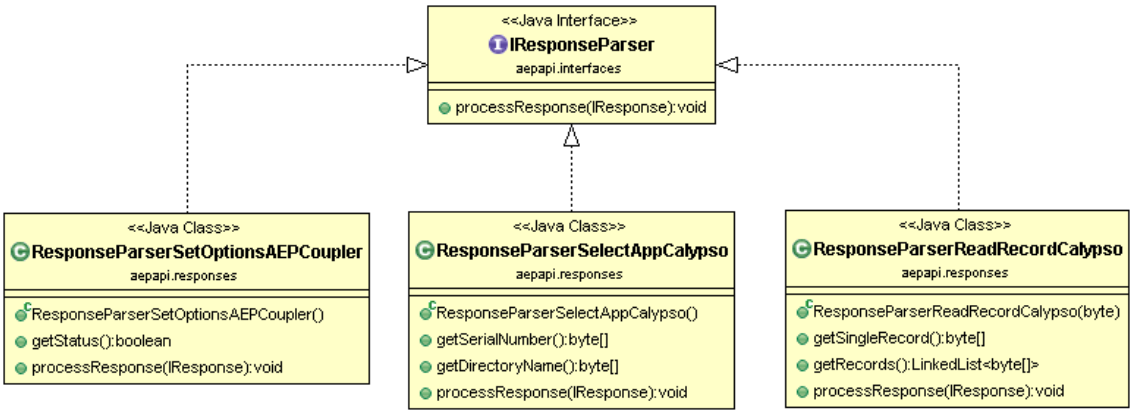

Figura 19 – *Response parsers* para os comandos utilizados no processo de leitura.

O *response parser* é responsável por verificar se o comando enviado foi corretamente executado, sendo lançada uma exceção caso isso não aconteça.

Cada *response parser* é também responsável por verificar se a resposta que está a interpretar corresponde ao comando esperado. Esta verificação evita que seja utilizado um *response parser* numa resposta a um comando diferente do comando para o qual o *response parser* foi desenhado.

4.7 Comandos do Leitor e do *Smart Card Calypso*

Nesta secção são abordados os comandos implementados para interagir tanto com leitor como com o *smart card*. Os comandos implementados são utilizados pelo validador *Android* para obter a informação necessária, a partir do *smart card*, para o processo de validação.

Dado o objetivo do trabalho, foram implementados os comandos necessários para que seja possível ler a informação necessária que se encontra armazenada nos *smart cards Calypso*, nomeadamente os contratos e o número de série do próprio *smart card*, permitindo assim desenvolver o validador *Android*.

Cada um destes comandos é independente da forma de comunicação utilizada com o leitor. Ou seja, não é necessário, realizar alterações a nenhuma das classes que contêm os métodos e os dados referentes a cada um destes comandos, se se pretender mudar o dispositivo de execução, por exemplo de um PC para um *tablet* com SO *Android*. Mesmo que estejam a ser utilizadas duas formas diferentes de comunicar com a porta COM (uma para o PC outra para o *tablet Android*), a única alteração necessária continua ser a implementação de um novo *IOManager* e indicar à classe que representa o leitor (*Coupler*) para utilizar esse novo *IOManager*.

De seguida são abordados os comandos implementados, para o leitor e para os *smart cards*, juntamente com a descrição do que cada comando permite realizar. Alguns destes comandos são específicos para o leitor utilizado, sendo que os comandos para o *smart card Calypso* são independentes do leitor em causa.

4.7.1 Comando *Get Software Version*

O comando *get software version* é utilizado para pedir ao leitor que indique a versão corrente do *software* nele instalado.

Este comando não recebe parâmetros e a resposta pode ser interpretada como uma *string* que contém as informações relativas à versão do *software* instalado no leitor.

A resposta a este comando depende apenas do leitor e pode ser utilizada pela aplicação como um teste para verificar se a comunicação com o leitor está a funcionar corretamente. Caso a aplicação não receba a resposta a este comando significa que a comunicação com o leitor não está a funcionar corretamente.

4.7.2 Comando *Set Options*

O leitor é um dispositivo portátil que é alimentado por uma bateria, sendo que está por isso configurado para entrar em modo de poupança de energia e até mesmo desligar-se, após estar algum tempo sem utilização. Desta forma o estado gerado pelos

comandos já enviados pode ser perdido, sendo necessário repetir o envio dos comandos previamente enviados.

Esta situação pode ser problemática, quando pode existir um longo período de inatividade, entre comandos, devido por exemplo a existência de latência na rede, ou até mesmo quando a aplicação está a ser executada em *debug* e se encontra parada durante um longo período de tempo num *break point*.

De forma a evitar que nestas situações o leitor entre em modo de poupança de energia ou se desligue foi implementado um comando que permite alterar as opões de energia do leitor.

O comando implementado recebe como parâmetro dois booleanos, que indicam se o leitor pode ou não entrar em modo de poupança de energia, ou desligar-se quando não é utilizado.

O *response parser* responsável por processar a resposta a este comando apenas verifica se o comando foi executado ou não com sucesso, permitindo assim aplicação saber do sucesso da execução do comando.

4.7.3 Comando *Search Card*

O comando *search card* é utilizado para que o leitor ligue a antena e posteriormente detete e comunique com um cartão. Este comando pode ser parametrizado com um *timeout* que é utilizado pelo leitor e que tem um valor no máximo de 2550 milissegundos. Se durante o decorrer deste tempo não for apresentado um cartão, o leitor responde ao comando indicando que não foi detetado nenhum cartão.

É necessário indicar ao leitor que tipos de cartões o mesmo deve procurar. No âmbito deste projeto os únicos cartões detetados são os *smart cards Calypso*, ou seja, se for aproximado ao leitor um cartão que não seja do tipo *Calypso*, esse cartão é ignorado.

Caso seja detetado um cartão, o leitor responde ao comando enviando a informação que identifica o tipo do cartão juntamente com alguns dados que foram gerados pelo mesmo.

O *response parser* que interpreta a resposta ao comando *search card* permite obter um objeto, que representa o cartão e os dados que o mesmo enviou através do leitor.

4.7.4 Comando *Antenna Off*

Tal como foi referido anteriormente a antena é ligada com o comando *search card*, desta forma é necessário ter o cuidado de desligar a antena assim que termina a interação com o cartão, com vista a poupar a bateria do leitor.

Para isso foi implementado o comando *antenna off* que quando enviado desativa a antena do leitor, desligando assim a ligação ao cartão.

Para comunicar novamente com o cartão, após executar este comando, é necessário executar de novo o comando *search card*.

4.7.5 Comando *Select Application*

Este comando permite selecionar uma *application* através do identificador da *application* pretendida. No âmbito deste projeto a única *application* selecionada foi a *Calypso application*.

Para executar este comando é necessário enviar os *bytes* que identificam este comando juntamente com os *bytes* que identificam a *application* pretendida. É também necessário que tenha sido previamente enviado o comando *search card.* Caso este procedimento não seja feito o leitor responde com um erro devido à não execução prévia do comando *search card*.

Após a execução deste comando com sucesso, a *application* cujo identificador foi indicado no comando passa a ser a *application* corrente, e pode ser utilizada pelos comandos enviados posteriormente sem que seja indicado novamente o identificador da *application* em causa.

Este comando é um comando APDU, ou seja, é um comando que é enviado diretamente para o *smart card*, sendo apenas necessário indicar os *bytes* que são enviados para o *smart card*, ou seja, não é necessário adicionar *bytes* de *header* e *trailer*. Isto acontece porque a própria classe que permite enviar comandos para o *smart card* adiciona automaticamente os *bytes* necessários para os comandos APDU.

Na resposta a este comando são recebidas várias informações relacionadas com a *application*, tal como o número de série. Os dados recebidos na resposta a este comando estão no formato TLD.

4.7.6 Comando *Read Record*

Como já foi dito anteriormente a informação presente no cartão está armazenada em ficheiros e três dos quatro tipos de ficheiros existentes guardam a informação na forma de *records*.

Foi portanto necessário implementar um comando que permitisse ler *records*. Para ler um *record* é necessário parametrizar o comando com o identificador do ficheiro que se pretende ler. Este identificador pode ser de quatro tipos diferentes, o primeiro consiste em não indicar o identificador do ficheiro que se pretende ler, fazendo com que o ficheiro utilizado seja o ficheiro corrente e com que seja lido apenas um *record*. O segundo tipo é semelhante ao primeiro, no entanto podem ser lidos mais que um *record*.

O terceiro e o quarto tipo são semelhantes aos dois primeiros tipos, sendo que a única diferença reside no facto de ser necessário indicar especificamente qual o ficheiro que se pretende ler através do respetivo SFI.

Após envio deste comando para o cartão e caso o mesmo tenha sido executado com sucesso, são enviados na resposta os *bytes* presentes no ou nos *records* solicitados no comando.

Tal como o *select application*, este comando é um *IsoCommand*, ou seja, é comando direto para o cartão.

O *parser* da resposta necessita de saber qual o tipo de pedido que foi utilizado, para interpretar a resposta no formato correto, sendo que no caso de ser lido apenas um *record* os *bytes* vêm diretamente na resposta e que quando se pretende ler mais do que um registo, os dados vêm na forma do TLD.

Antes de enviar este comando para o cartão é necessário enviar o comando *serch card* para o leitor e o comando *select aplication* para que seja selecionada a *application* que contém os ficheiros a que se pretende aceder.

4.8 Problemas no *Software* do Adaptador de *Bluetooth* na API *Android*

No decorrer do desenvolvimento da biblioteca de interação com leitores e *smart cards* verificou-se a ocorrência de um problema na utilização do adaptador de *Bluetooth* fornecido pela API (*Application Programming Interface*) do SO *Android.*

O problema verificado impede o estabelecimento da ligação com o leitor *Bluetooth*, sendo lançado uma exceção pela API *Android* que indica que "o *socket* poderia ter sido fechado". Sempre que este erro ocorre é difícil estabelecer a ligação com o leitor sem desligar e ligar novamente o *Bluetooth*. Após a pesquisa *online* da mensagem de erro evidenciou-se que esta exceção se deve a um erro no SO *Android*, que já foi reportado no *bug tracker* do SO *Android*, mas ainda não foi corrigido.

Para contornar este problema foi implementado um método que desliga e liga o *Bluetooth* sempre que é detetado este problema e volta a tentar detetar o leitor.

Caso após cinco tentativas não tenha sido possível estabelecer a ligação com o leitor, o utilizador é informado, podendo neste caso repetir o processo.

Visto este problema só se evidenciar antes do leitor ser detetado, não afeta o bom funcionamento da biblioteca implementada após a correta deteção do leitor.

5 Validador Desenvolvido para o SO *Android*

Neste capítulo é abordado o validador *Android*[6] juntamente com as funções que o mesmo desempenha no processo de validação, interação com o leitor (interface externa) e com a camada que liga ao serviço de validação (ver Figura 20).

Este capítulo está dividido em 8 secções: 5.1 onde é descrito de forma genérica a aplicação de validação desenvolvida para o *tablet* com SO *Android*; 5.2 onde é descrita a camada de apresentação e respetivos ecrãs; 5.3 onde é descrito como é usada aplicação e são indicados quais os passos para iniciar o processo de validação; 5.4 é descrito o mecanismo utilizado para que seja detetado um cartão junto ao leitor, é este mecanismo que é responsável por iniciar o processo de validação; 5.5 é descrito o processo de validação que é iniciado assim que é detetado um cartão junto ao leitor; 5.6 é descrito o mecanismo de *heart beat*; 5.7 onde é abordado o mecanismo de segurança utilizado na comunicação com o serviço de validação; 5.8 onde é descrito o comportando da aplicação de validação perante falhas que possam ocorrer durante o processo de validação.

Figura 20 – Arquitetura do dispositivo *Android*.

[6] Consiste num *tablet* com SO *Android* que permite verificar se um determinado bilhete tem um contrato válido.

5.1 Aplicação de Validação

A aplicação *Android* de validação consiste num demonstrador que utiliza a biblioteca de acesso a *smart cards* implementada, no âmbito deste projeto, para interagir com o leitor e os respetivos *smart cards*.

Este demonstrador é um "*thin device*" e destina-se a revisores que pretendam verificar se o título de transporte de um passageiro é válido, sendo que a validação do título de transporte é feita por um serviço *web*. A aplicação de validação desenvolvida para o SO *Android* inclui ainda uma camada de apresentação, onde é mostrado o resultado da validação do título de transporte em causa.

5.2 Camada de Apresentação

A camada de apresentação é constituída por duas *activities*: a primeira permite que o revisor introduza os seus dados e verifique o estado da ligação entre o *tablet* com SO *Android* e o leitor, a segunda *activity* mostra se um título de transporte é ou não válido.

A Figura 21 ilustra o primeiro ecrã a ser mostrado ao revisor, onde o mesmo deverá introduzir as suas credenciais para poder utilizar a aplicação.

Figura 21 – Ecrã de *login*.

A Figura 22 mostra os três ecrãs da interface de utilizador da *activity* de validação.

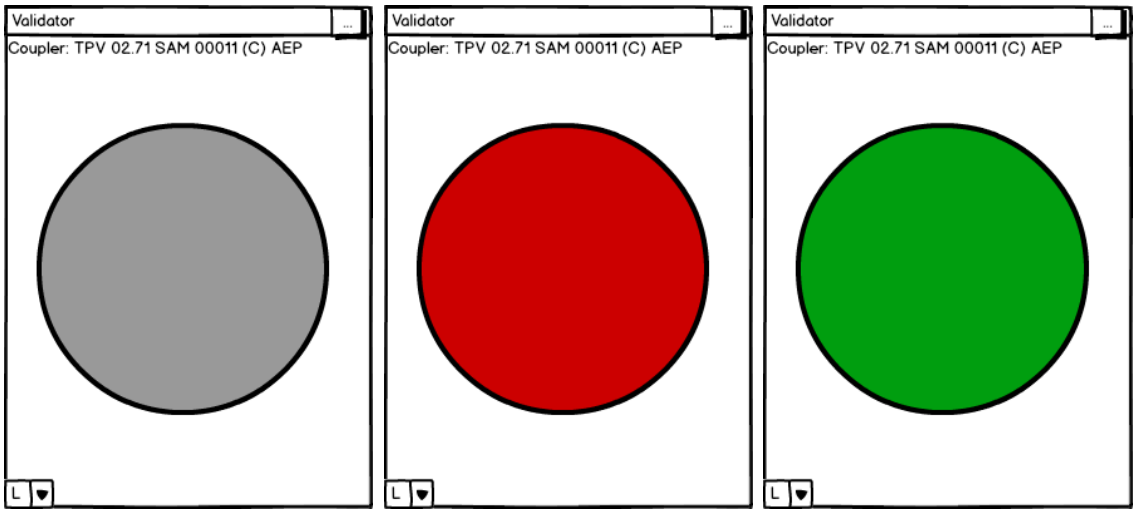

Figura 22 – *Mockups* dos três ecrãs de validação.

Na Figura 22, pode observar-se que a imagem mais à esquerda consiste no ecrã que é mostrado enquanto não é apresentado nenhum cartão. A segunda imagem mostra o ecrã que é mostrado quando é apresentado um cartão inválido junto ao leitor e o terceiro ecrã é mostrado quando é apresentado um cartão válido.

5.3 Utilizar a Aplicação

Na primeira *activity* são mostradas as *textboxs* onde o revisor deverá escrever o seu nome de utilizador e a sua *password*. É necessário aguardar que a ligação ao leitor seja realizada, sendo que até lá o botão "*continue*" encontra-se inativo. Assim que o leitor é detetado com sucesso o botão "*continue"* fica ativo e pode ser premido pelo revisor para que seja mostrada a interface de validação.

Assim que é iniciada a interface de validação é verificado se o nome de utilizador e *password* introduzidos são válidos. Em caso afirmativo é iniciado o processo de validação, sendo que para isso a aplicação *Android* pede ao leitor que detete um cartão utilizando o comando *search card* e caso a resposta indique que foi detetado um cartão é então iniciado o processo de validação dos contratos presentes no cartão. Este processo é repetido através de um mecanismo de *polling*.

5.4 Mecanismo de *Polling*

Um leitor nunca toma a iniciativa de iniciar a comunicação com o dispositivo a que está ligado (neste caso um *tablet* com SO *Android*). Desta forma para que o *tablet* com SO

Android seja informado da presença de um cartão junto ao leitor, é necessário implementar um mecanismo de *polling* onde periodicamente é perguntado ao leitor se o mesmo deteta algum cartão.

Para o mecanismo de *polling* foi implementada uma classe que executa uma tarefa, de forma contínua e repetida até que a tarefa executada diga que o *polling* deve parar. Esta situação ocorre tipicamente quando a tarefa foi finalizada com sucesso ou quando houve uma exceção. No caso da tarefa de *polling* usada para detetar um cartão, o *polling* termina quando for detetado um cartão.

Quando o *polling* termina é chamada uma função de *callback*, associada à tarefa, que recebe como parâmetro o resultado da execução da tarefa. No caso da procura de cartão o resultado da tarefa é o objeto que representa o cartão.

Este mecanismo de *polling* pode ser parado de forma permanente (quando a *activity* termina) ou temporária (quando a *activity* deixa de ficar visível). É também possível retomar a execução de uma tarefa (quando a *activity* volta a ficar visível).

Com vista a melhorar a eficiência as tarefas são executadas por *threads* do *threadpool*, com o mecanismo de *cached threads*, desta forma o número de *threads* existente é constante ao longo da execução das tarefas.

Para abortar a execução de uma tarefa, é interrompida a *thread* que se encontra atualmente a executar a tarefa. De forma a libertar a *thread* do *threadpool* o fio de execução da *thread* em causa termina. Assim que se pretende retomar o *polling* é apenas necessário submeter uma nova tarefa ao *threadpool*.

5.5 Processo de Validação

O processo de validação é mostrado de forma genérica pelo fluxograma da Figura 23.

Inicialmente a aplicação *Android* utiliza o mecanismo de *polling* para detetar se foi aproximado um cartão do leitor e em caso afirmativo é iniciado o processo de validação.

De seguida no processo de validação é utilizado o comando *set options* para desativar o modo de poupança de energia do leitor, garantindo assim que o mesmo não entra em modo de poupança energia durante a interação com cartão.

Seguidamente é utilizado o comando *select application* para selecionar a *Calypso application*, pois é nesta *application* que está o ficheiro "*Contracts*" (onde estão armazenados os contratos). Se este comando for realizado com sucesso a *Calypso application* passa a ser a *application* corrente. É também na resposta a este comando que é obtido o número de série do cartão.

É então necessário ler a informação presente no ficheiro "*Contrats*" para obter os contratos existentes. Este ficheiro é constituído por quatro *records*, sendo que cada *record* corresponde a um contrato.

São então lidos todos os contratos, um a um, utilizando o comando *read record* passando como parâmetro a este comando o *byte* que identifica o ficheiro que guarda os contratos. Por cada contrato lido é feito um pedido validação ao serviço *web* em que o mesmo responde indicando se o contrato é válido ou não. No mesmo pedido é também enviado o número de série do cartão e a zona, uma vez que o mesmo contrato pode ser válido numa ou mais zonas.

Posteriormente é analisado o resultado de cada pedido enviado ao serviço de validação e caso uma das respostas seja válida é mostrado no ecrã do *tablet* com SO *Android* um círculo verde durante um segundo e meio. Caso nenhum dos contratos seja válido é mostrado um círculo vermelho durante um segundo e meio.

Finalmente é enviado um comando para o leitor com o objetivo de ativar novamente o mecanismo de poupança de energia em caso de inatividade.

Por fim é mostrado o círculo cinzento e é reativado o mecanismo de *polling*, o que permite que seja novamente repetido o processo para um cartão que seja aproximado do leitor.

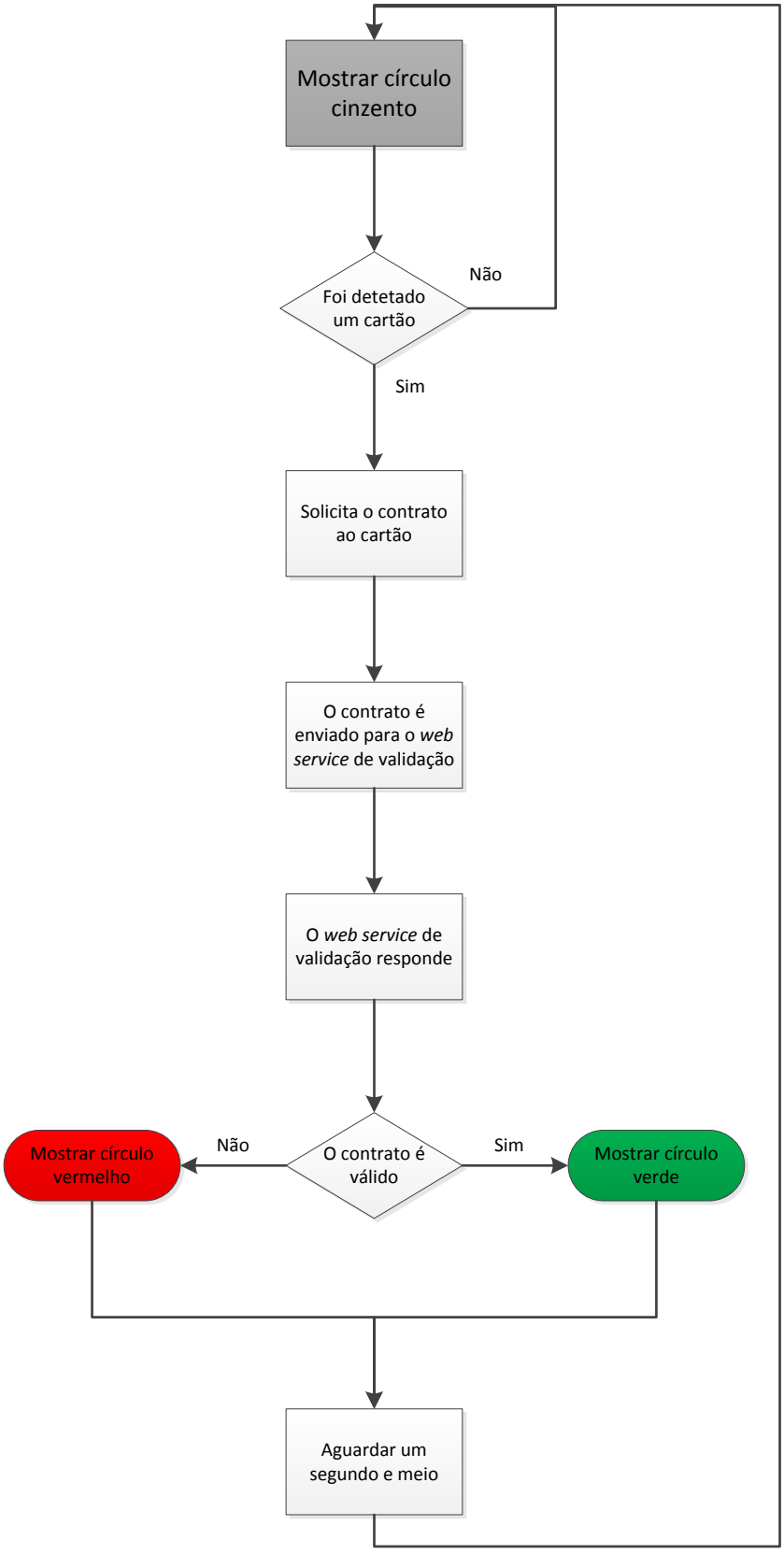

Figura 23 – Fluxograma do processo de validação.

5.6 *Heart Beat*

De trinta em trinta segundos é enviada uma mensagem de *heart beat* para o serviço de validação. Esta mensagem serve apenas para indicar ao serviço de validação que o *tablet* com SO *Android* se encontra em funcionamento.

O primeiro *heart beat* é utilizado também para verificar se o nome de utilizador e a *password* introduzidos pelo revisor estão ou não corretos. Para isso, o *tablet* com SO *Android* fica a aguardar a resposta enviada pelo serviço e verifica se essa mesma resposta contém o código que indica que as credenciais são inválidas.

5.7 Segurança na Ligação ao Serviço de validação

Uma vez que para comunicar com o serviço de validação é necessário enviar o nome de utilizador e a respetiva *password* é necessário que seja garantida a confidencialidade destes dados. Para garantir a confidencialidade foi utilizado um certificado auto assinado do lado do servidor e foi utilizado o protocolo HTTPS (*Hypertext Transfer Protocol Secure*).

A aplicação *Android* suporta as duas formas de comunicação (segura: HTTPS e não segura: HTTP (*Hypertext Transfer Protocol*)) com serviço de validação.

5.8 Tratamento de Falhas de Comunicação entre o Validador *Android* e o Serviço de Validação

Nos pedidos enviados ao serviço de validação podem ocorrer alguns erros, quer de comunicação, quer proveniente do próprio serviço. Os erros provenientes do próprio serviço são erros que indicam que não foi possível realizar a operação pedida, assinalando na resposta o motivo.

No caso da validação de um contrato existem dois erros relevantes, o primeiro indica que o nome de utilizador e *password* do revisor estão incorretos. Quando este erro acontece o validador *Android* informa o utilizador de que o nome de utilizador e a *password* estão incorretos, terminando de forma graciosa a aplicação.

O segundo erro relevante pode ocorrer quando não existe comunicação entre o validador *Android* e o serviço de validação. Quando este erro ocorre, é mostrada uma

mensagem ao utilizador através de um *toast* que indica que o processo de validação não pode ser concluído uma vez que não é possível comunicar com o serviço de validação. Para fazer uma nova tentativa de validação basta aproximar novamente o *smart card* do leitor, e o validador *Android* tentará novamente estabelecer a comunicação com o serviço de validação por forma a realizar a validação do contrato presente no *smart card*.

6 Camada de Lógica de Validação (Serviço de Validação)

Neste capítulo é descrito o serviço *web* de validação implementado, abordando os métodos remotos desenvolvidos, o mecanismo de segurança envolvido na comunicação entre o serviço e os vários validadores *Android* e por fim o mecanismo utilizado para atualizar as estruturas de dados necessárias para o processo de validação (ver Figura 24).

O capítulo corrente encontra-se dividido em 3 secções: 6.1 descrição breve das funcionalidades do serviço de validação; 6.2 são explicadas as funcionalidades de cada método remoto implementado pelo serviço: (1) método que permite a um validador *Android* enviar *heart beats*; (2) método que permite que um validador *Android* solicite, ao serviço de validação, as configurações que vai usar e (3) método usado por um validador *Android* para verificar se um contrato é válido; 6.3 onde é descrito método de segurança usado para garantir a confidencialidade; 6.4 mecanismo utilizado para atualizar as estruturas de dados utilizadas para validação de um título de transporte; 6.5 é justificada a opção de deixar a biblioteca de interação com leitores e *smart cards* do lado do validador *Android* em vez de estar no lado do serviço de validação; 6.6 são abordados os diferentes tipos de nuvem computacional existentes; 6.7 são abordadas as características da nuvem computacional da *Amazon* juntamente com a implementação do serviço de validação nesta nuvem.

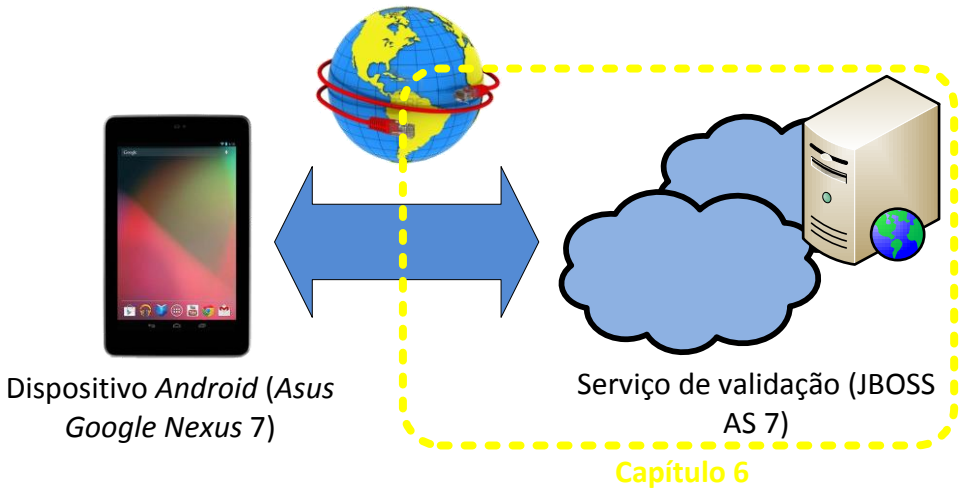

Figura 24 – Arquitetura do serviço de validação.

6.1 Implementação do Serviço de Validação

Como já foi dito, o serviço de validação implementado é um serviço *web* e permite validar um título de transporte. Para que tal seja possível é necessário que o *tablet* com SO *Android* comunique com este serviço *web,* enviando um ou mais contratos presentes no *smart card*, recebendo de seguida uma resposta que indica se cada um dos contratos é ou não válido. Por sua vez o serviço *web* implementado permite ainda que um terminal, neste caso o *tablet* com SO *Android*, receba as configurações que indicam as zonas possíveis para validação e o identificador do operador.

Como já foi previamente dito, o *tablet* com SO *Android* implementa a lógica "*thin device*", ou seja, é um dispositivo que não tem a lógica de validação armazenada localmente. Desta forma o *tablet* precisa de comunicar com o serviço de validação para verificar se um título de transporte é válido e para receber as configurações necessárias ao seu funcionamento.

6.2 Métodos Implementados

Foi portanto implementado um serviço REST (*Representational State Transfer*) que é utilizado pelo *tablet* com SO *Android* e que disponibiliza três operações: enviar *heart beat*, validar contrato (presente num título de transporte) e obter configurações. O serviço implementado é *stateless*, sendo por isso necessário, ao *tablet*, enviar as credenciais (nome de utilizador e *password*) em cada pedido realizado ao serviço.

Este serviço está alojado num servidor aplicacional JBOSS AS7 (*JavaBeans Open Source Software Application Server*) e o protocolo de comunicação utilizado é o protocolo HTTP ou o protocolo HTTPS.

Cada método deste serviço é disponibilizado num caminho (URL (*Uniform Resource Locator*)) diferente, recebe os parâmetros através da *query string* e responde enviando um objeto no formato JSON (*JavaScript Object Notation*). Cada resposta contém um código, quer seja uma resposta que indica sucesso na chamada ao método, quer seja uma resposta que indica a ocorrência de um erro. O objeto enviado na resposta contém ainda informação sobre a forma de texto, que é utilizada para descrever de forma breve a resposta ao método que o validador *Android* pretende executar. Esta

informação pode ser utilizada para, por exemplo, mostrar ao utilizador qual o motivo da exceção que impediu a correta execução do método no lado do serviço.

Em todos os métodos implementados, é verificado se o nome de utilizador e *password* estão corretos, e caso não se verifique que os mesmos são válidos é enviado na resposta para o validador Android, um objeto que indica que a operação não foi realizada devido à utilização de credenciais inválidas.

6.2.1 Enviar *Heart Beat*

Este método remoto permite que o *tablet* com SO *Android* envie uma mensagem que simula um "batimento cardíaco" para o serviço de validação. Neste método é apenas verificado se o utilizador introduziu as credenciais corretas e em caso afirmativo é enviada a resposta que indica que o *heart beat* foi realizado com sucesso.

Este método é utilizado apenas para informar o serviço de validação de que o *tablet Android* ainda se encontra em funcionamento.

6.2.2 Obter Configurações

De forma a tornar os dispositivos "*thin devices*" mais flexíveis, as configurações que os mesmos utilizam são obtidas a partir do serviço de validação. Para isso, é necessário que o cliente (neste caso o *tablet Android*) envie um pedido para o URL deste método remoto, com o nome de utilizador e *password*. Após verificar que as credenciais enviadas pelo cliente são válidas, são enviadas na resposta as zonas válidas e o identificador do operador correspondente ao nome de utilizador enviado pelo cliente.

6.2.3 Validar um Contrato

Este método permite que seja validado um título de transporte de um passageiro. Para isso, o serviço recebe um pedido, proveniente do *tablet* com SO *Android*, onde para além do nome de utilizador e *password* do revisor, é envida informação presente no *smart card*.

A informação enviada para validação é o número de série do *smart card*, o contrato presente no mesmo cartão e ainda a zona onde o passageiro se encontra.

Assim que o pedido chega ao serviço é verificado se o título de transporte é válido através do contrato recebido, sendo para isso verificado se o contrato é igual (*byte* a *byte*) a um contrato presente na lista de contratos válidos para o número de série do cartão em causa. Esta verificação é feita por um método da classe validador, que é também responsável por verificar se o contrato é válido para a zona em causa. Desta forma é possível que um contrato seja válido numa ou mais zonas.

No final é enviado para o validador *Android* que fez o pedido de validação uma resposta, no formato JSON, que indica se o título de transporte é ou não válido.

6.3 Segurança na Comunicação com o Serviço

A comunicação entre o serviço implementado e os vários validadores *Android* é realizada através da rede pública *Internet*. Visto que as *passwords*, os números de série e os contratos são enviadas em claro na *query string* de cada pedido, é necessário garantir a confidencialidade na comunicação com o serviço.

Para garantir a confidencialidade foi utilizado o protocolo HTTPS, sendo que para isso foi necessário criar um certificado auto assinado, cujas chaves são utilizadas para impedir que os dados transferidos entre os dois intervenientes da comunicação sejam acedidos por terceiros.

6.4 Atualização dos Dados no Serviço

Os objetos criados quando um validador *Android* realiza um pedido ao serviço de validação, são destruídos automaticamente sempre que o serviço se encontra sem receber pedidos durante um determinado período de tempo.

Desta forma sempre que é recebido um pedido, pode ser necessário obter os dados e criar as estruturas de dados utilizadas para, por exemplo, realizar uma validação de um contrato.

Com o objetivo de diminuir o *overhead* de obter as estruturas de dados necessárias para atender um pedido, foi implementado o mecanismo de *schedule* presente no JBOSS AS7. Este mecanismo é utilizado então para agendar a chamada a um método que vai atualizar as estruturas de dados que são utilizadas para atender pedidos.

As estruturas de dados são então guardadas num objeto que não é destruído, sendo por isso garantido que os dados permanecem acessíveis, mesmo que o serviço esteja um longo período de tempo sem receber pedidos.

Desta forma quando chega um novo pedido ao serviço de validação, este utiliza as estruturas de dados existentes em vez de construir novamente essas mesmas estruturas, diminuindo assim o *overhead* necessário para atender um pedido.

6.5 Serviço de Validação a Controlar a Interface Externa

Um dos objetivos iniciais no contexto do projeto corrente, pretendia desenvolver um dispositivo "*thin device*" em que a lógica de negócio estaria no serviço de validação. Uma das vantagens da lógica de negócio estar centralizada prende-se com o facto de diminuir o tempo e o custo das alterações realizadas. A diminuição de custo e de tempo acontece, porque as alterações à lógica de negócio são feitas unicamente no serviço de validação e não manualmente em todos os dispositivos.

Outra vantagem seria uma maior simplicidade na lógica presente na aplicação de validação desenvolvida para o *tablet* com SO *Android* uma vez que a complexidade estaria no serviço de validação. Seria também mais fácil adaptar o sistema a novos leitores e *smart cards* sem ser necessário fazer alterações na aplicação.

Desta forma poderia ser correto pensar que se deveria colocar a biblioteca de interação com leitores e *smart cards* no lado do serviço de validação em vez de estar no lado da aplicação desenvolvida para o *tablet* com SO *Android*. De facto as vantagens apresentadas anteriormente verificar-se-iam, no entanto seria necessário passar através da rede um número elevado de *bytes* referentes aos comandos a enviar para o leitor ou *smart card*, e receber as respetivas respostas. Ou seja, verificar-se-ia uma maior latência no processo de validação, o que poderia ser crítico se for necessário validar um grande número de títulos no menor espaço de tempo possível.

Desta forma optou-se por deixar a biblioteca de interação com leitores e *smart cards* do lado da aplicação de validação desenvolvida para *tablet* com o SO *Android*, em vez de estar centralizada no serviço de validação.

6.6 Camada de Lógica de Validação na Nuvem

Nesta secção é abordada a nuvem computacional e a integração da mesma com o projeto desenvolvido, a fim de validar o conceito de "*thin device*" numa logica de PaaS, implementada na *Amazon Web Services*.

Esta aproximação do desenvolvimento usando a nuvem computacional terá vantagens numa fase de implementação do projeto em diferentes clientes, porque evita a compra de *hardware* dedicado.

O crescente interesse comercial da nuvem deve-se à existência de uma maior tendência na utilização de *outsourcing* na área das TI (tecnologias de informação) o que permite diminuir o número de recursos humanos necessários para a manutenção dessas mesmas infraestruturas. Outra vantagem prende-se com um baixo custo inicial, o que facilita a entrada de novos fornecedores de serviços.

Nesta fase do projeto SmartCITIES esta componente está a ser aborda noutros projetos a decorrem no ISEL, usando uma aproximação de SaaS: (1) projeto "A Bilhética dos Transportes de Passageiros na Nuvem Computacional" [14], a decorrer, no qual se esta a usar a nuvem computacional da Google; (2) o projeto "Estudo do Paradigma Computação em Nuvem" [15], concluído, no qual se usou a nuvem computacional da Microsoft, o *Windows Azure*. Isto fez com que o presente trabalho fosse orientado para a nuvem computacional da *Amazon*, utilizando uma aproximação PaaS, uma vez que a empresa Link, se encontra com software já implementado numa lógica de serviços, sendo portanto mais simples adaptar os seus serviços já existentes ao serviço PaaS da *Amazon*.

6.6.1 Computação na Nuvem

O modelo de negócio da nuvem computacional baseia-se normalmente em formas de pagamento por utilização, onde é pré-definida uma capacidade máxima sendo posteriormente calculado o custo real dos recursos usados. Sendo assim, existem inúmeras vantagens para os negócios novos ou para aqueles cujo crescimento é incerto. Neste modelo os custos iniciais são mínimos garantindo qualidade e

elasticidade. Num passado recente, esta realidade não seria possível sem investir em *data centers* privados.

A computação na nuvem permite desta forma que sejam disponibilizadas infraestruturas de alto desempenho para soluções na área das tecnologias de informação. Estas infraestruturas podem ser utilizadas para alojar vários serviços [16].

Um dos riscos associados à nuvem computacional prende-se com o facto de que os serviços são disponibilizados através da *Internet*, ou seja, é necessário ter um cuidado acrescido na comunicação com a nuvem computacional e ainda proteger os dados nela armazenados.

Existe também um risco de falha na disponibilidade que pode ocorrer por uma falha de rede ou falha da infraestrutura da nuvem computacional devido a um erro ou a um ataque (por exemplo DOS (*Denial of Service*)).

Estes exemplos permitem demostrar a importância que a segurança tem na nuvem computacional.

É difícil definir a nuvem computacional, pois a mesma está relacionada com um grande conjunto de tecnologias [17]. No âmbito da publicidade ao modelo de negócio a nuvem computacional é referida como um conjunto de soluções que facilitam a partilha de recursos computacionais.

Outra definição comum consiste em definir a nuvem computacional, como o acesso a um conjunto de recursos e/ou serviços pagos por utilização, que têm uma elasticidade ilimitada e instantânea. Desta definição surgem então três formas de utilizar a nuvem computacional: IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*).

6.6.2 Modelos de serviço de nuvem computacional

Neste subcapítulo são abordados os diferentes modelos de nuvem computacional que podem ser disponibilizados sob a forma de serviços.

6.6.2.1 Infraestrutura como Serviço (IaaS)

Infraestrutura como serviço é a camada de mais baixo nível e fornece serviços de suporte básico de infraestrutura e refere-se à partilha dos recursos de *hardware* geralmente utilizando uma tecnologia de virtualização. Os recursos podem facilmente ser aumentados dependendo das necessidades. Tipicamente, os recursos são máquinas virtuais que têm associada uma gestão para controlar a criação e manutenção, evitando igualmente o acesso não controlado à informação.

6.6.2.2 Plataforma como Serviço (PaaS)

Plataforma como serviço é a camada intermédia que oferece serviços ao nível da plataforma e proporciona um ambiente para alojamento de aplicações específicas. Genericamente esta camada disponibiliza um ambiente de execução de *software*. Por exemplo, um servidor PaaS permite instalar aplicações *web* sem ser necessário adquirir e configurar servidores reais.

O modelo PaaS garante a proteção e armazenamento de dados num serviço, no entanto os dados guardados nesta nuvem computacional, necessitam de proteção adicional (por exemplo: cifra) para garantir a sua confidencialidade.

A distinção entre os serviços PaaS e IaaS nem sempre é clara, sendo por isso frequente a mesma empresa fornecer os dois.

6.6.2.3 *Software* como Serviço (SaaS)

Software como serviço é a camada de mais alto nível e disponibiliza o acesso a uma aplicação completa oferecida ou paga como serviço [18]. O SaaS garante o alojamento completo de aplicações na nuvem computacional, eliminando desta forma a necessidade de instalar e executar a aplicação localmente no computador ou num servidor local, reduzindo assim os custos de manutenção de *software*.

6.7 Amazon Web Services

O AWS (*Amazon Web Service*) é um serviço de nuvem computacional que é disponibilizado pela *Amazon* [19]. A *Amazon*, não se iniciou na área de tecnologia, mas sim na área de venda *online*. No entanto utilizou, mais tarde, os seus *data centers*

espalhados pelos vários continentes para disponibilizar um serviço de nuvem computacional através da *Internet*, que consiste num serviço PaaS.

O acesso aos serviços disponibilizados pela *Amazon* pode ser realizado através de uma API REST ou de uma API SOAP. A infraestrutura disponibilizada pela *Amazon* garante a segurança no acesso aos serviços disponibilizados no AWS, sendo que em cada pedido realizado é necessário enviar as credências para aceder ao serviço em causa, garantindo assim o acesso apenas a utilizadores autorizados. Todas as mensagens enviadas são assinadas com o algoritmo HMAC (*Hash Message Authentication Code*).

O AWS pode ser utilizado para implementar algumas funcionalidades utilizadas pelo serviço de validação referido nos subcapítulos anteriores.

6.7.1 *Amazon Simple Storage Service*

É possível armazenar ficheiros na *Amazon* através do S3 (*Amazon Simple Storage Service*), sendo que cada ficheiro armazenado fica acessível através da de um URL. Existem três tipos de entidades diferentes no S3, nomeadamente objeto, chaves e *bucket* [20]. Cada ficheiro armazenado neste serviço é identificado por uma chave, e está armazenado num *bucket*. Um *bucket* guarda um conjunto ficheiros, sendo portanto análogo a uma pasta. Um ficheiro dentro de um *bucket* é denominado de objeto.

6.7.2 *Amazon Elastic Compute Cloud*

O Ec2 (*Amazon Elastic Compute Cloud*) é serviço *web* que disponibiliza uma máquina virtual, que pode ser utilizada para alojar outros serviços [21]. Esta máquina virtual pode ser utilizada de forma semelhante a uma máquina física, sendo possível aceder à máquina através de ambiente remoto. O ficheiro que contém a imagem máquina virtual é denominado de AMI (*Amazon Machine Images*) e está armazenado no serviço S3.

Uma aplicação que esteja a correr no Ec2 pode ser replicada por várias máquinas, sendo garantido o balanceamento de carga entre todas as instâncias. Uma outra vantagem deste tipo de redundância é o facto de ser garantida tolerância a falhas, ou

seja, se uma máquina deixar de funcionar corretamente, os pedidos são redirecionados para outra máquina.

6.7.3 *Amazon Machine Images*

Como já foi dito anteriormente, a AMI (*Amazon Machine Images*) consiste numa imagem de uma máquina virtual e contém toda a informação necessária ao arranque e funcionamento dessa máquina [21]. Todas as AMI estão encriptadas e são armazenadas no serviço S3 sobe a forma de um ficheiro.

Uma AMI, tal como uma máquina física, contém um sistema operativo e ainda *software* que pode ser, por exemplo, um motor de base de dados ou um servidor HTTP.

É o ficheiro de AMI que é replicado e utilizado por todas as instâncias de máquinas virtuais que estão a correr no Ec2.

6.7.4 Camada da Lógica de Validação na *Amazon Web Services*

A nuvem computacional disponibilizada pela *Amazon* permite criar um ambiente de execução semelhante ao ambiente utilizado para correr o serviço de validação. O serviço de validação encontra-se a correr em ambiente *Windows* num servidor aplicacional JBOSS.

Para implementar o serviço de validação na nuvem computacional da *Amazon* foi necessário utilizar uma imagem AMI que contém o *Windows server 2008* pré-instalado. Nesta máquina foi instalado o *software* necessário para correr o servidor aplicacional JBOSS, tendo sido previamente instalado a plataforma *Java*. A instalação foi realizada através do ambiente de trabalho remoto.

Após a instalação foi realizado o *deploy* do serviço de validação para a máquina virtual presente na nuvem computacional da *Amazon*, ou seja, foi adicionado o serviço de validação ao servidor JBOSS presente na máquina virtual.

7 Arquitetura *Multi-Tenancy*

Neste capítulo são abordadas as vantagens da utilização de uma arquitetura *multi-tenancy*, e como a arquitetura do projeto corrente pode ser adaptada a diferentes cenários. As descrições das alterações necessárias em cada cenário são apresentadas segundo a ordem dos capítulos 4, 5 e 6 presentes neste documento.

De forma a suportar uma arquitetura *multi-tenancy* que inclui serviços é relevante conhecer os vários operadores de transporte envolvidos numa dada região. É necessário ter em conta que numa região os vários operadores de transporte podem ter passageiros e títulos de transporte em comum, ou seja, é necessário ter uma visão global sobre a lógica de negócio que possa ser comum a todos os operadores.

Assim, surge um modelo com múltiplas áreas (por exemplo, área da região metropolitana de Lisboa ou área da região metropolitana do Porto) e cada área é constituída por um ou mais operadores e contém alguma da lógica de negócio em comum a todos os operadores (ver Figura 25).

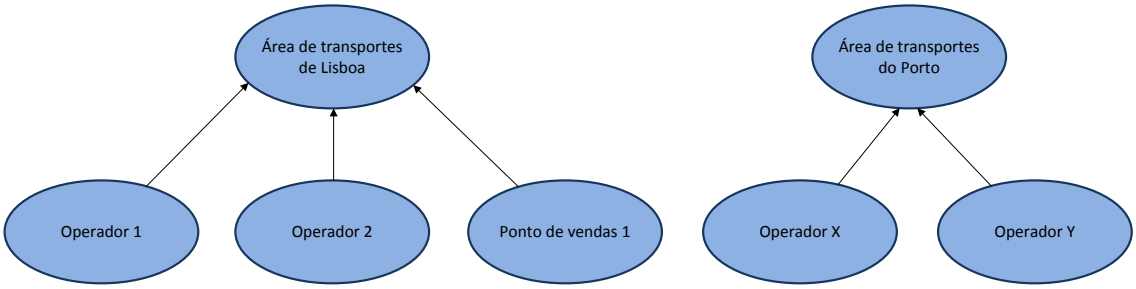

Figura 25 – Estrutura hirarquica *multi-tenancy* simples.

Desta forma existem duas camadas, a superior (área) onde estão as informações comuns a todos os operadores e as camadas inferiores onde estão os operadores que acedem aos dados armazenados na camada superior.

Com este modelo pode ser necessário colocar a informação de cada operador numa base de dados comum. Esta situação pode originar problemas de privacidade, segurança e extensibilidade. Existe portanto o risco de um operador conseguir aceder a informação de um outro operador concorrente. Acrescenta-se ainda o facto de que

cada sistema de bilhética de cada operador tem especificidades que podem divergir dos restantes operadores.

Um segundo modelo consiste na utilização de um sistema que contém várias bases de dados separadas (nas várias camadas inferiores), uma por cada operador, juntamente com uma base de dados central (no nível superior). É necessário ter em conta que a informação é sempre produzida pelos níveis inferiores que são responsáveis por enviar a informação que desejam partilhar para os níveis superiores.

7.1 Aplicação Prática para a Área Metropolitana de Lisboa

O projeto corrente, como já foi dito, consiste num *tablet* com SO *Android* a correr uma aplicação de validação, que comunica com um serviço *web* para obter as configurações e realizar a validação de títulos.

Pode-se então fazer a analogia com a situação real da área metropolitana de Lisboa, onde cada revisor de cada operador teria um nome de utilizador e uma *password* que usaria para fazer *login* no seu *tablet* com SO *Android*.

As configurações como as zonas e o identificador de cada operador estariam guardadas no serviço de validação, sendo enviadas para o validador *Android* assim que revisor faz *login* com sucesso. Quando os revisores fazem *login* ficam automaticamente com o *tablet* com SO *Android* configurado para fazer validação pelo operador a que pertencem.

Os números de série e os contratos dos títulos que são válidos deverão estar também armazenados no serviço de validação, permitindo assim que, por exemplo, um validador do Metro consiga validar um título Lisboa Viva Carris/Metro que foi emitido pela Carris quando o passageiro carregou o seu passe.

Para adaptar a aplicação de validação desenvolvida para o *tablet* com SO *Android* no âmbito deste projeto ao cenário da área de Lisboa (excluindo os cartões 7 Colinas) não seria necessário fazer alterações. Isto acontece porque o código da biblioteca de interação com leitores e *smart cards*, foi desenhado e testado com os cartões Lisboa Viva, que são os cartões utilizados na área de metropolitana de Lisboa.

No entanto, seria necessário fazer algumas alterações para que os cartões 7 Colinas fossem validados, uma vez que estes usam a tecnologia CTS 512 [22] que é diferente da tecnologia *Calypso*. Inicialmente seria necessário fazer uma alteração ao comando *search card* para fazer com que o leitor detetasse os dois tipos de cartões (7 Colinas e Lisboa Viva). Após fazer esta alteração deveriam ser implementados e utilizados os diferentes comandos para obter a informação necessária presente no cartão 7 Colinas e envia-la para o serviço *web* onde iria ser realizada a validação.

7.2 Alterações Necessárias para a Utilização na Área Metropolitana do Porto

A utilização de *smart cards* e de leitores pode variar de área metropolitana para área metropolitana. Desta forma pode ser necessário fazer alterações à lógica de negócio e ao código de interação com os leitores e *smart cards*, para adaptar o projeto desenvolvido a uma área metropolitana cujas especificidades sejam diferentes das especificidades com que o projeto corrente foi desenvolvido.

Neste subcapítulo são abordadas as alterações necessárias para adaptar o projeto desenvolvido à área metropolitana do Porto.

7.2.1 Alterações à Interface Externa

Uma vantagem da arquitetura usada no contexto deste projeto é o facto do *software* implementado para *Android* ter também uma arquitetura dividida em camadas. Desta forma é mais fácil adaptar o código implementado a novos cartões, ou ainda a diferentes tipos de leitores, sendo necessário fazer poucas alterações para além das alterações à camada de *software* que lida com a interface externa.

Assim se na área metropolitana do Porto for utilizado um tipo de leitor diferente, é necessário adaptar a biblioteca de interação implementada no contexto do projeto corrente ao novo leitor. Seria portanto necessário enviar os comandos para o leitor de acordo com o formato que o novo leitor consegue interpretar, ou seja, era necessário implementar uma nova classe de *Coupler* (subtipo da interface *ICoupler*).

Nesta classe que iria representar o novo leitor e que permitiria enviar comandos para o respetivo leitor, os comandos enviados para o leitor deveriam ser encapsulados com

o *header* e o *trailer* específico para o leitor utilizado na área metropolitana do Porto. Seria também necessário utilizar o algoritmo para gerar o código CRC específico para o novo leitor.

É também necessário implementar o código que permite que sejam enviados comandos APDU através do novo leitor. É portanto necessário, preparar esta nova classe *Coupler* para encapsular corretamente os *bytes* necessários para identificar um comando APDU e fazer com que o leitor o reencaminhe para o *smart card*. É também necessário que os *bytes* da resposta sejam extraídos, para que posteriormente sejam transmitidos ao *response parser* que os vai interpretar.

Se o leitor utilizado na área metropolitana do Porto fosse o mesmo leitor que foi utilizado no âmbito deste projeto, não seria necessário realizar quaisquer alterações ao código.

7.2.1.1 Alterações Realizadas aos Comandos Implementados

Uma vez que os comandos enviados para o leitor poderiam ser diferentes dos comandos utilizados pelo leitor usado no contexto deste projeto, poderia ser necessário implementar os comandos necessários ao processo de validação, de forma a poderem ser enviados e executados no novo leitor.

Destes comandos destaca-se o comando *search card* e o comando *antenna off*. No comando *search card* é necessário implementar o comando que indique ao leitor que deve iniciar o processo de procura de um cartão e comando *antenna off* deverá ser um comando que permita ao leitor poupar recursos terminando a comunicação com o cartão.

7.2.1.2 Response Parsers

Para os comandos implementados especificamente para o novo leitor seria também necessário implementar o respetivo *response parser*, para que seja possível obter o resultado da execução do comando, ou ainda que seja indicado ao programador o erro que ocorreu na execução de um comando.

7.2.1.3 APDU

Como já foi dito anteriormente este tipo de comandos é independente do leitor, desta forma se o *smart card* utilizado na área metropolitana do Porto (cartão Andante) for o mesmo *smart card* que é utilizado na área metropolitana de Lisboa (cartão Lisboa Viva), não é necessário realizar qualquer alteração a este tipo de comandos e aos respetivos *response parsers*.

No entanto se for utilizado um *smart card* diferente é preciso implementar os comandos necessários para obter a informação utilizada no processo de validação, como o número de série e os contratos presentes no cartão, ou outra informação equivalente que possa ser enviada para o serviço *web* onde é feita a validação.

7.2.2 Alterações Realizadas ao Validador *Android*

Para que o validador *Android* funcione corretamente seria apenas necessário, utilizar a biblioteca correta, se necessário, com as alterações referidas anteriormente.

Caso o leitor utilizado fosse diferente seria necessário utilizar os novos comandos para procurar o cartão e desligar a antena.

Caso o *smart card* utilizado não fosse o cartão Lisboa Viva seria necessário obter as informações necessárias ao processo de validação, como o número de série e os contratos presentes no *smart card* ou informação equivalente, e reencaminha-los para o serviço de validação.

Seria ainda necessário utilizar um SAM do lado leitor e ler o contrato presente no *smart card* através de uma sessão segura, para assim garantir a autenticidade dos dados presentes no cartão em causa. Desta forma seria necessário implementar também os comandos que permitem abrir e fechar uma sessão segura.

7.2.3 Alterações Realizadas ao Serviço de Validação

Para o processo de validação pode ser necessário fazer alterações ao serviço de validação *online*. No caso de ser utilizado o cartão *Calypso*, não eram necessárias quaisquer alterações, bastaria apenas guardar neste serviço *web* os números de série e os contratos válidos, para todos os operadores da área metropolitana do Porto, e

indicar quais as possíveis zonas de validação que são utilizadas pelos revisores para verificar se um título de transporte é válido.

No entanto se fosse utilizado um novo tipo de *smart card* seria necessário suportar a forma de validação para esse cartão, sendo portanto necessário fazer alterações às estruturas de dados e aos métodos utilizados nesse processo. Ou seja, seria necessário guardar a informação no lado do serviço de validação, que permitisse identificar se um cartão contém um título de transporte válido, e alterar o método que verifica a validade de forma a utilizar essa informação no processo de validação.

7.3 Alterações Necessárias para Adaptar o Projeto a uma Cancela do Metro

A arquitetura implementada no contexto deste projeto permite desenvolver dispositivos mais flexíveis. Neste capítulo são enunciadas as alterações necessárias ao projeto corrente para implementar uma cancela do metro, sendo de salientar que algumas componentes desenvolvidas poderão ser reaproveitadas. É assumido que os sistemas do metro em causa utilizam os títulos Lisboa Viva (tecnologia *Calypso*) e os títulos 7 Colinas/Viva Viagem (tecnologia CTS 512).

7.3.1 Alterações à Interface Externa

Uma vez que esta biblioteca foi desenvolvida e testada com os *smart cards Calypso* não é necessário realizar qualquer alteração para os cartões Lisboa Viva.

No entanto, tal como foi dito anteriormente, seria necessário implementar os comandos necessários para obter a informação utilizada no processo de validação dos cartões 7 Colinas. Seria também necessário alterar o comando *search card*, para que fossem pesquisados os tipos de cartões Lisboa Viva e 7 Colinas.

O restante código desta biblioteca poderá ser utilizado na cancela sem qualquer alteração, sendo no entanto necessário informar a cancela do resultado da validação, para fazer com que a mesma abra ou não de acordo com a validade do título apresentado pelo passageiro.

7.3.2 Alterações Realizadas ao Validador *Android*

Na aplicação de validação desenvolvida para o *tablet* com SO *Android*, seria inicialmente necessário adaptar a aplicação de forma a utilizar a nova biblioteca de interação com leitores e *smart cards*. Esta alteração é necessária para que seja possível enviar os comandos para interação com o cartão 7 Colinas, caso seja este o cartão apresentado junto ao leitor. Para o caso do cartão Lisboa Viva, não era necessário qualquer alteração.

Nesta aplicação seria também necessário acrescentar o código que permite controlar a cancela. Uma solução seria criar uma biblioteca, que permitisse abrir e fechar a cancela, ou ainda aceder aos sensores para saber, por exemplo, se o passageiro já passou a cancela.

Seria ainda necessário alterar a interface de utilizador, de forma a mostrar alguma informação extra como o prazo de validade do título introduzido, ou ainda o número de viagens ou saldo restante no título de transporte.

7.3.3 Alterações Realizadas ao Serviço de Validação

No serviço de validação era necessário implementar as estruturas de dados que iriam suportar o novo cartão 7 Colinas.

Seria também necessário ter em conta o novo tipo de cartão (7 Colinas), para que de acordo com o cartão (Lisboa Viva ou 7 Colinas) utilizar um processo de validação diferente. No caso do cartão Lisboa Viva o processo de validação é o processo descrito no capítulo 6, sem qualquer alteração.

Seria também necessário aplicar as medidas referidas no subcapítulo 7.2.2 com objetivo de garantir a autenticidade dos dados presentes no cartão Lisboa Viva. Seria também necessário descodificar o contrato enviado pelo validador *Android* e através do modelo de dados do contrato confirmar se o passageiro pode entrar ou não entrar na estação. Para verificar se o contrato é válido é necessário aplicar as regras específicas que o operador em causa utiliza, que incluiu, por exemplo, indicar se um título de transporte é validado por um determinado tempo ou num determinado conjunto de zonas.

No caso do cartão 7 Colinas seria necessário usar a informação como o número de série, e verificar se a viagem corrente do passageiro cujo revisor está a validar se encontra válida. Uma vez que uma viagem no título 7 Colinas é válida durante uma hora é também necessário neste caso verificar a data e a hora atuais e comparar com a data e a hora da última viagem cobrada ao passageiro. Se a diferença for superior a uma hora o título é considerado inválido.

7.4 Processo de Venda de Títulos de Transporte

Para carregar um cartão Lisboa Viva é necessário atualizar o contrato presente no *smart card*, indicando um novo prazo de validade para o cartão, juntamente com a indicação em que operadores o contrato é válido.

O cartão Lisboa Viva utiliza um *smart card Calypso*, que apenas aceita que sejam escritos dados, se os mesmos forem escritos no contexto de uma sessão segura entre o leitor e o *smart card* em causa. Assim é necessário implementar alguns comandos e utilizar um SAM do lado do leitor.

É de salientar que é necessário abrir uma sessão segura porque, apesar de ser possível ler informação dos *smart cards Calypso* sem qualquer autenticação, não é possível escrever no *smart card* sem que o leitor esteja autenticado através da utilização de um SAM.

Para abrir uma sessão segura é necessário utilizar um SAM, pois é o SAM que contém as chaves utilizadas para a sessão segura. Na sessão segura é autenticado tanto o SAM como o *smart card Calypso*, juntamente com toda a informação enviada por ambos.

Para que fosse possível realizar um carregamento deveriam portanto ser implementados os comandos que permitem abrir uma ligação segura através do SAM, e no final terminar essa sessão segura. Deveria ainda ser implementando o comando *write record* que permite que seja escrito um *record* num dado ficheiro.

Uma vez que o processo de venda envolve uma transação bancária é necessário, utilizar um serviço de pagamentos. A implementação deste serviço encontra-se fora do âmbito deste projeto, no entanto para realizar o processo de venda num ambiente de

produção seria necessário utilizar o serviço de pagamentos. Este serviço seria utilizado no pagamento de cada título de transporte adquirido ou recarregado.

7.4.1 Atualização do Contrato Presente no *Smart Card*

Para escrever o novo contrato é necessário abrir uma sessão segura com o *smart card*. Assim o primeiro comando a ser utilizado é o *open secure session* que inicia uma sessão segura com o *smart card*.

De seguida seria necessário utilizar o comando *write record* que vai permitir que o contrato presente no *smart card* seja atualizado para um novo contrato válido. Este comando permite portanto realizar uma operação de escrita num ficheiro, sendo que neste caso o ficheiro onde seria realizada a escrita era o ficheiro denominado "*Contracts*" presente na *Calypso application*. O novo contrato seria escrito no mesmo *record* onde estava armazenado o contrato anterior. Assim os restantes *records* do ficheiro "*Contracts*" podem ser utilizados por outras entidades emissoras de títulos.

Após a realização das alterações deve ser fechada a comunicação com o comando *close secure session*, sendo que se este comando for realizado com sucesso as alterações realizadas com o comando de escrita são tornadas permanentes. No caso de ocorrer um erro na execução deste comando, as alterações realizadas no contexto desta sessão são automaticamente revertidas. Um dos motivos para que este comando não seja realizado com sucesso pode ser a não autenticação com sucesso de uma das partes (*smart card Calypso* ou SAM).

Um só contrato pode ser válido em mais do que um operador, por exemplo um contrato pode ser válido, na Carris e no Metro. Sendo assim para realizar uma venda é necessário conhecer também formato do contrato que deve ser escrito no *smart card Calypso* (presente no cartão Lisboa Viva). Para a região de Lisboa este formato é especificado pela OTLIS.

8 Testes Realizados

São descritos neste capítulo os testes efetuados a todo o sistema, para avaliar o grau de satisfação relativamente aos objetivos pré-definidos. Os testes dividem-se em dois grupos distintos: testes de integração e testes de performance. É também descrito todo o ambiente a nível de *hardware* e *software* utilizado durante a execução dos mesmos.

8.1 Ambiente de Realização de Testes

Os testes decorreram em ambiente controlado com *hardware* escolhido propositadamente para o efeito. Não foi possível efetuar testes em ambiente de produção, visto que isso implicaria a aquisição de diverso material para distribuir por vários utilizadores.

Assim os testes foram realizados num ambiente que incluía o *hardware* referido anteriormente (leitor AEP e um *tablet Asus Google Nexus 7* com *Android* 4.2.1) juntamente com um PC portátil onde está a correr o servidor de validação com JBOSS AS 7 em ambiente *Windows 7 Home Premium*. Os cartões de teste consistiram em cartões Lisboa Viva, carregados previamente com contratos válidos.

Foram ainda realizados testes em duas máquinas virtuais alojadas na nuvem computacional da *Amazon*, sendo que uma das máquinas encontra-se na área dos Estados Unidos (em Oregon) e outra encontra na área da Europa (em Irlanda).

8.2 Preparação para a Execução dos Testes

Inicialmente é necessário ler o contrato presente nos cartões que se deseja marcar como válidos e adiciona-los ao ficheiro que guarda os contratos válidos e que está presente no serviço de validação. É necessário também indicar as zonas para as quais cada contrato é válido.

De seguida é necessário criar uma conta que é utilizada pelo revisor, indicando qual o operador a que o revisor pertence.

8.3 Testes de Integração

Os testes de integração representam todos os testes efetuados para garantir o correto funcionamento de cada um dos módulos em conjunto com todos os outros. Para uma correta avaliação de cada módulo foram definidos diversos cenários, onde se apresentam nesta secção os mais críticos para o correto funcionamento de todo o sistema desenvolvido.

8.3.1 Teste de Deteção do Leitor

Ao fazer o teste de deteção de leitor foram verificados ocasionalmente alguns problemas no *tablet* com SO *Android*. Estes problemas devem-se à própria API do *Android* que não funciona corretamente em algumas situações (ver subcapítulo 4.8), indicando que não é possível abrir a ligação *Bluetooth* ao leitor AEP. Desta forma o tempo de deteção do leitor pode variar de teste para teste, uma vez que pode ser necessário repetir o processo de deteção do leitor quando o mesmo falha.

Verificou-se também que uma vez conseguida a ligação entre o *tablet* com SO *Android* e leitor, ambos se mantêm sempre em funcionamento correto sem qualquer falha ou problema na ligação *Bluetooth*.

8.3.2 Teste de Início de Sessão (*Login*)

Verificou-se que a aplicação de validação (validador Android) realiza o login corretamente, informando o utilizador caso as credenciais introduzidas não sejam válidas. Verificou-se também que quando o revisor inicia sessão com sucesso recebe as configurações corretas para o seu operador, e estas são usadas no processo de validação.

8.3.3 Teste de Validação

Após o leitor ser detetado com sucesso, o *tablet* com SO *Android* encontra-se pronto a ser utilizado. Após introduzir as suas credenciais na *main activity* o revisor pode então premir o botão "*continue*". De seguida é mostrada a *activity* onde se pode realizar a validação.

Nesta *activity* o revisor prime o botão "*Get Coupler Info*" e o *tablet Android* fica pronto a ser utilizado, sendo portanto possível validar títulos a partir deste momento.

Inicialmente verificou-se que quando o revisor inicia sessão com sucesso são apresentadas as possíveis zonas de validação.

Verificou-se também que quando era apresentado um título de transporte válido, era mostrado o círculo verde (ver Figura 26), mas apenas para as zonas em que o mesmo está autorizado a circular. Ou seja, apesar de o título ser válido numa zona não é mostrado o círculo verde se o mesmo estiver numa outra zona onde o seu título é inválido. Nesta circunstância é então mostrado o círculo vermelho.

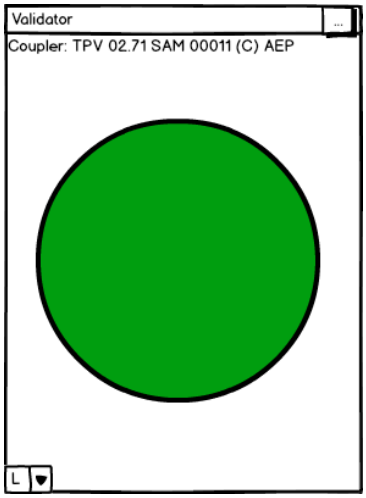

Figura 26 – Ecrã que indica que o título de transporte é válido.

Verificou-se também que ao ser apresentado junto do leitor um título desconhecido, ou um título cujo contrato já expirou (foi removido do ficheiro de contratos válidos) é sempre mostrado o círculo vermelho (ver Figura 27).

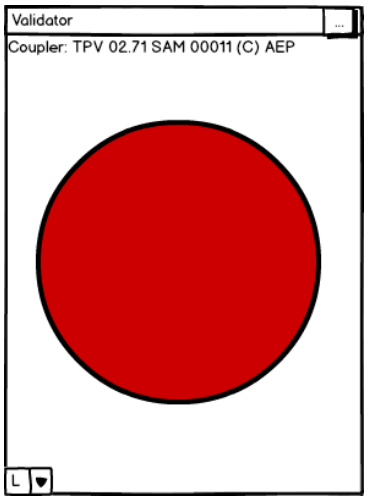

Figura 27 – Ecrã que indica que o título de transporte não é válido.

Verificou-se também que a aplicação tolera de forma graciosa a remoção do *smart card* sem que toda a informação necessária ao processo de validação tenha sido lida, ou ainda quando ocorre um erro devido a uma falha de comunicação com o serviço de validação.

8.4 Testes Metro/Carris

Os títulos de transporte presentes no cartão Lisboa Viva permitem que um contrato seja válido em um ou mais operadores, desta forma um passageiro com o mesmo contrato pode andar por exemplo nos autocarros da Carris e no Metro.

A arquitetura da solução implementada suporta que um contrato seja válido em mais do que um operador, sendo apenas necessário indicar no lado do serviço de validação que o contrato é válido para mais do que um operador de transporte (indicando para que operadores o contrato é válido).

8.5 Testes de Desempenho

Como já foi dito anteriormente é necessário validar os quatro contratos presentes no *smart card*. Visto que a comunicação com o *smart card* e com o serviço de validação podem ser operações demoradas, é relevante ter em conta o tempo que demoram estas operações no processo de validação. O *tablet* com SO *Android* estava ligado à *Internet* através de uma rede Wi-Fi a operar com a norma 802.11g, que permite uma velocidade de transmissão de dados até 54 Mbps. A informação transmitida nesta rede é protegida pelo protocolo de segurança WPA.

Para testar a performance do sistema desenvolvido, foram realizados vários pedidos de validação sendo "anotado" o tempo total que decorre no processo de validação desde que o cartão é detetado junto ao leitor, até ao momento em que o *tablet* com SO *Android* mostra o resultado da validação. Em cada pedido de validação são enviados sete comandos para o leitor, sendo que cinco desses comandos são APDU reencaminhados pelo leitor para o *smart card*.

Foram realizados três testes diferentes, sendo que para cada teste o serviço de validação estava alojado numa máquina diferente: a primeira máquina consistia num

servidor PC na rede local, a segunda e terceira máquina encontravam-se alojadas na nuvem computacional da *Amazon*.

8.5.1 Teste com o serviço de validação a correr no PC

O primeiro teste de performance foi realizado com o serviço de validação a correr no PC descrito no subcapítulo 8.1, ou seja, foi colocado o serviço *web* de validação num PC na rede local da empresa Link e foi usado o validador *Android* desenvolvido para testar o processo de leitura de cartões e o processo de validação *online*. Após a realização de seis validações verificaram-se os resultados ilustrados pela Tabela 2. Cada uma das entradas é referente um dia da semana, de segunda-feira a sábado e resulta da média dos resultados dos vários testes realizados ao longo do dia.

Tabela 2 – Tempos de validação com serviço de validação alojado no PC.

	Acesso ao *smart card* (ms)	Acesso ao serviço de validação (ms)	Total (ms)	Total (s)
	867	1693	2560	2,6
	831	1665	2496	2,5
	931	1741	2672	2,7
	967	1722	2689	2,7
	908	2139	3047	3,0
	969	2254	3223	3,2
Média	912	1869	2781	2,8
Pior Tempo	969	2254	3223	3,2
Melhor Tempo	831	1665	2496	2,5

O tempo calculado é o resultado da soma dos tempos de duas operações principais: o processo de leitura dos dados presentes no *smart card* e a validação desses mesmos dados no serviço *web*. Estes tempos foram calculados de forma separada.

Ao realizar este teste verificou-se que em média um pedido de validação demora cerca de 2,8 segundos, sendo gasto cerca de um segundo na interação com o *smart card*, e 1,8 segundos na interação com o serviço web de validação.

8.5.2 Testes com o serviço de validação a correr na *Amazon*

Foram realizados dois testes com o serviço de validação a correr em duas máquinas virtuais distintas alojadas no serviço Ec2 da *Amazon*.

O primeiro teste consistiu em utilizar uma máquina virtual localizada na área dos Estados Unidos. Ao realizar os seis testes de validação verificaram-se os resultados ilustrados pela Tabela 3.

Tabela 3 – Tempos de validação com serviço de validação alojado na *Amazon* (Estados Unidos).

	Acesso ao *smart card* (ms)	**Acesso ao serviço de validação (ms)**	**Total (ms)**	**Total (s)**
	1008	4301	5309	5,3
	1033	2751	3784	3,8
	1081	2214	3295	3,3
	1019	4607	5626	5,6
	1261	3925	5186	5,2
	924	2171	3095	3,1
Média	1054	3328	4383	4,4
Pior Tempo	1261	4607	5626	5,6
Melhor Tempo	924	2171	3095	3,1

Dos resultados obtidos, verificou-se que em média o tempo de acesso ao *smart card* não varia em relação ao teste anterior, demorando cerca de um segundo. Isto acontece porque não é necessária a interação com o serviço de validação até que a informação seja totalmente lida do *smart card*. No entanto, o tempo de validação no serviço alojado na *Amazon* demorou cerca de três segundos. Este aumento no tempo de resposta, por parte do serviço de validação, em relação ao teste anterior (em que é utilizado um PC) deve-se à distância geográfica da máquina virtual e o *tablet* com SO *Android*.

O segundo teste feito na *Amazon* consistiu em utilizar uma máquina virtual na área da Europa. No decorrer deste teste foram também realizadas seis validações, e obtiveram-se os resultados indicados pela Tabela 4.

Ao analisar os resultados verificou-se que o tempo de resposta por parte do serviço de validação diminui em relação ao teste anterior, para cerca de dois segundos, sendo que o tempo total de validação é de cerca de 3,4 segundos.

Tabela 4 – Tempos de validação com serviço de validação alojado na *Amazon* (Europa).

	Acesso ao *smart card* (ms)	Acesso ao serviço de validação (ms)	Total (ms)	Total (s)
	940	2551	3491	3,5
	962	2723	3685	3,7
	926	1787	2713	2,7
	975	3111	4086	4,1
	1031	2634	3665	3,7
	1012	1824	2836	2,8
Média	974	2438	3413	3,4
Pior Tempo	1031	3111	4086	4,1
Melhor Tempo	926	1787	2713	2,7

A Tabela 5 mostra os resultados obtidos nos três testes realizados. Ao analisar e comparar estes resultados pode concluir-se que existe uma latência, que se verifica essencialmente no processo de validação *online*.

Tabela 5 - Comparação entre os resultados dos três testes de validação.

	Teste com PC (ms)	Teste na Amazon (Estados Unidos) (ms)	Teste na Amazon (Europa) (ms)
Média	2781	4383	3413
Pior Tempo	3223	5626	4086
Melhor Tempo	2496	3095	2713

Para um validador esta latência de 3,4 segundos não é um problema, uma vez que este tipo de operação não é realizada aos passageiros frequentemente. No entanto verifica-se que é um valor de latência elevado, que poderia ser problemático se fosse utilizado numa cancela automática, uma vez que o passageiro teria de aguardar este tempo para poder entrar e sair da estação.

Esta latência verifica-se devido essencialmente ao tempo de transmissão do pedido de validação através da rede, uma vez que são realizados quatro pedidos de validação, um por cada contrato existente no *smart card Calypso*. Há fatores condicionantes que podem ser melhorados com o objetivo de otimização do processo de validação como, por exemplo, utilizar a tecnologia 3G como meio de comunicação no *tablet* com SO

Android e enviar todos os contratos presentes no *smart card* no mesmo pedido de validação. Desta forma o tempo do pedido de validação seria mais reduzido.

9 Conclusão

No início do trabalho foram identificadas as funcionalidades críticas a implementar para atingir o objetivo proposto neste projeto. É necessário provar a possibilidade da implementação do conceito "*thin device"* proposto no projeto SmartCITIES *Cloud Ticketing* da empresa Link. Para o efeito e tendo em conta as perspetivas de negócio da empresa Link, optou-se pela implementação num *tablet* com a plataforma *Android*. Tendo em consideração a diversidade de operações existentes na bilhética optou-se pela criação de um validador de títulos de transporte cuja interface externa (biblioteca de interação com leitores e *smart cards*) está a ser utilizada em sistemas que estão em produção e que são mantidos pela empresa Link no projeto SmartCITIES *Cloud Ticketing*.

Foi também identificado que o sistema teria de ser suficientemente rápido para funcionar corretamente em situações de grande fluxo de pessoas (grupos acima de 50 pessoas) e verificou-se que o sistema responde num período de tempo adequado ao processo de validação (cerca 3,4 segundos).

A utilização da nuvem computacional pode ser um problema, se for utilizada noutros tipos de dispositivos, como uma cancela automática, onde o tempo decorrido no processo de validação poderia obrigar o passageiro a ter de aguardar demasiado tempo para entrar ou sair da estação. Existe ainda o problema de ser necessária uma ligação constante ao serviço de validação, uma vez que caso a comunicação falhe não é possível validar títulos de transporte. Neste contexto, existe o risco de um passageiro legítimo não poder utilizar o transporte.

Para evitar este problema, o serviço de validação pode ser replicado na rede local de cada estação diminuindo assim a probabilidade de falha, ou seja, se ocorresse uma falha seria localizada apenas numa estação, permitindo que as restantes estações funcionem normalmente.

Este trabalho serviu de base para a publicação de um artigo em conferência internacional:

- João C. Ferreira, Porfírio Filipe, Gonçalo Cunha, João Silva. *Cloud Terminals for Ticketing System* disponível nos *proceedings* da 5th *International Conferences on Advanced Service Computing*, *service computation* 2013, Maio 27 - Junho 1, 2013 - Valencia, Espanha.

Com base neste trabalho foi ainda submetido o seguinte artigo para a conferência internacional:

- Agostinho Baía, João C. Ferreira, Porfírio Filipe, Gonçalo Cunha. *ANDROID AS A CLOUD TICKET VALIDATOR*, submetido para a *International Conference on Cloud & UBiquitous Computing & Emerging Technologies* (CUBE 2013) a realizar em Pune (India) de 15 a 16 de Novembro de 2013.

Este trabalho contribuiu para o projeto QREN da Link, SmartCITIES ao desenvolver um protótipo num *tablet* com SO *Android*, permitindo assim validar a proposta do conceito de "*thin device*".

De igual forma foi, a título pessoal, muito positivo o período (Janeiro 2013 a Junho 2013) que estive na empresa Link, o qual permitiu adquirir conhecimentos na área da bilhética e perceber os mecanismos de interação com os leitores, *smart cards*, quais os comandos envolvidos no processo de leitura de um contrato, onde e como estão armazenados os contratos, e como obter a informação utilizada no processo de validação (número de série e os contratos).

Em relação à proposta de *multi-tenancy* embora não seja um dos objetivos do projeto corrente, o presente trabalho permitiu dar os primeiros passos provando que o conceito pode ser implementado. No entanto, a latência das comunicações pode ser um problema para operações de grande fluxo de pessoas, como são as portas de entradas em transportes públicos. Foi provado ainda que, com a arquitetura *multi-tenancy,* é mais fácil implementar dispositivos mais flexíveis reutilizando lógica comum aos vários tipos de dispositivos.

9.1 Trabalho Futuro

Apesar de não serem objetivos do projeto corrente são enumerados de seguida alguns pontos que ficaram em aberto.

Uma alteração a realizar futuramente seria validar o contrato utilizando uma sessão segura entre o leitor e o *smart card*, garantido desta forma que a informação proveniente do *smart card* é autêntica. Neste caso, seria necessário implementar os comandos que permitem abrir e fechar uma sessão segura com o *smart card*. Seria também necessário utilizar um SAM do lado do leitor para autenticar os comandos enviados pelo leitor e verificar a autenticação das respostas enviadas pelo *smart card*. Desta forma seria garantido que o contrato enviado pelo *smart card* era autêntico. A utilização de segurança iria aumentar um pouco a latência, uma vez que seria necessário enviar os comandos para o SAM, para que os mesmos fossem assinados, e do lado do *smart card Calypso*, seria também necessário assinar todas as respostas enviadas.

No futuro poderia também ser acrescentada uma funcionalidade para permitir ao validador *Android* a validação de títulos emitidos em códigos QR (*Quick Response*). Neste tipo de títulos seria necessário interpretar os títulos a partir de uma imagem impressa numa folha de papel. Por exemplo, poderia ser utilizada a câmara do *tablet* com SO *Android* para fotografar o código QR impresso na folha e de seguida utilizar uma aplicação para obter a informação presente no código. De seguida seria necessário enviar essa informação para o serviço de validação, onde a mesma seria validada.

Uma outra tarefa referente a trabalho futuro seria desenvolver um novo validador noutro tipo de dispositivo móvel com outro SO. Assim, ao desenvolver a aplicação de validação noutros SO, como o iOS ou *Windows Phone*, aumentaria o conjunto de dispositivos que poderiam ser utilizados como validadores. Desta forma, seria ainda mais fácil escolher diferentes tipos de dispositivos, podendo por exemplo ter conta o custo mais reduzido ou o melhor desempenho.

Bibliografia

[1] Link Consulting SA, “http://www.link.pt/,” [Online].

[2] J. C. Ferreira, C. Gomes, P. Filipe, G. Cunha e J. Silva, “Taas – Ticketing as a Service, in proceedings of CLOSER 2013 - 3rd International Conference on Cloud Computing,” Aachen-Germany, 2013.

[3] ASK, “ASK, contactless smart cards, tickets, inlays and terminals,” [Online]. Available: http://www.ask-rfid.com/. [Acedido em 01 05 2012].

[4] J. Rebelo e J. Matias, “SmartCards,” 2013, projeto de Licenciatura (LEIC), a decorrer.

[5] F. M. v. U. Chaves, “O Telemóvel como Bilhete Electrónico,” 2009.

[6] R. F. A. R. Carapeto, “Segurança e Privacidade na Gestão Intermodal de Bagagens com RFID,” 2010.

[7] RFID Journal, “RFID Journal - FAQS - Q&A Section about RFID Uses & RFID Technology - RFID (Radio Frequency Identification) Technology News & Features,” 2012. [Online]. Available: http://www.rfidjournal.com/faq/18/. [Acedido em 18 Novembro 2012].

[8] A.-R. Sadeghi, I. Visconti e C. Wachsmann, “User Privacy in Transport Systems Based on RFID E-Tickets,” 2008.

[9] M. Mezgnani, “Study on electronic ticketing in public transport,” 2008.

[10] Calypso Networks Association, “CALYPSO Handbook,” 2010.

[11] Calypso Networks Association, “CALYPSO SPECIFICATION REV.3 Portable Object Application Version 3.1,” Spirtech, 2009.

[12] Joint Technical Committee ISO/IEC, "ISO/IEC 7816-3:1997/PDAM-2 Information technology - Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols," 2002.

[13] AEP Ticketing Solutions, "http://www.aep-italia.it/," [Online]. [Acedido em 1 Maio 2013].

[14] N. Maurício, "A Bilhética dos Transportes de Passageiros na Nuvem Computacional," 2013, projeto de Mestrado (MEIC), a decorrer.

[15] C. Gomes, "Estudo do Paradigma Computação em Nuvem," 2012, projeto de Mestrado (MEIC).

[16] R. Bhadauria, R. Chaki, N. Chaki e S. Sanyal, "A Survey on Security Issues in Cloud Computing," 2011.

[17] L. M. Vaquero, L. Rodero-Merino, J. Caceres e M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," 2009.

[18] A. T. Velte, T. J. Velte e R. Elsenpeter, Cloud Computing: A Pratical Approach, 2010.

[19] Amazon Web Services, "Getting Started with AWS," 2013.

[20] Amazon Web Services, "Amazon Simple Storage Service Guia de conceitos básicos API Version 2006-03-01," 2006.

[21] Amazon Web Services, "Amazon Elastic Compute Cloud User Guide API Version 2012-07-20," 2012.

[22] ASK, "C.ticket Range - Get in with the generation of intelligent ticketing," 2010.