INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Eletrónica

e Telecomunicações e de Computadores

A Bilhética dos Transportes de Passageiros na Nuvem Computacional

Nuno Miguel Negrão Maurício

(Licenciado)

Relatório de projeto realizado para obtenção do grau de Mestre

em Engenharia Informática e de Computadores

Orientador:

Doutor Porfírio Pena Filipe

Júri:

Presidente: Mestre Pedro Alexandre Seia Cunha Ribeiro Pereira

Vogais: Mestre Carlos Jorge de Sousa Gonçalves

Doutor Porfírio Pena Filipe

dezembro de 2013

Tipicamente, as empresas procuram estratégias que permitam expandir o seu negócio e que o aproximem cada vez mais dos seus clientes. Neste contexto, as aplicações móveis desempenham um papel emergente na condução das estratégias de negócio das empresas, devido às suas características de mobilidade, diversidade e facilidade de aquisição e utilização. Essencialmente, a mobilidade combinada com o acesso à Internet, apanágio dos dispositivos móveis, proporciona o aparecimento de novas oportunidades de negócio dinamizando atividades económicas e sociais.

Nesse âmbito, o presente trabalho pretende propor uma arquitetura para conceção de aplicações móveis, com lógica balanceada, ou seja, cujo comportamento pode residir parcialmente numa nuvem computacional.

Para efeitos de avaliação pretende-se aplicar a proposta a dispositivos, terminais no domínio da bilhética de transportes coletivos de passageiros, e implementar réplicas desses dispositivos sob a forma de aplicações móveis que interagem com a nuvem computacional.

Palavras-Chave

Computação em nuvem, *Software as a Service*, Aplicação móvel, Bilhética

Typically, companies seek strategies to expand your business and the increasingly closer to their customers. In this context, mobile applications play an emerging role in driving the business strategies of companies, due to its characteristics of mobility, diversity and ease of purchase and use. Essentially, mobility combined with access to the Internet, the prerogative of mobile devices, provides the appearance of new business opportunities, boosting economic and social activities.

In this context, this project intends to propose an architecture for mobile applications conception, balanced with logic, or whose behavior may lie partially in cloud computing.

For the purpose of evaluating, the proposal intends to apply it in devices, terminals in the field of ticketing for public transportation of passengers, and implement, in accordance with the proposal, replicas of these devices in the form of mobile applications that interact with cloud computing.

KEYWORDS

Cloud Computing, Software as a Service, Mobile Application Ticketing

Quero agradecer a todos os familiares e amigos que me apoiaram e me deram força para concluir este trabalho de mestrado.

Especialmente, eu quero agradecer ao Professor Doutor Porfírio Pena Filipe pela orientação muito bem conduzida e a Carina Nobre Gomes por todo o apoio motivacional, o qual foi fundamental.

API	*Application Programming Interface*
AA	*Appcelerator Alloy*
ACS	*Appcelerator Cloud Services*
ACW	*Android Callable Wrappers*
AT	*Appcelerator Titanium*
AWS	*Amazon Web Services*
Basic	*Beginner's All-purpose Symbolic Instruction Code*
BI	*Business Intelligence*
BLOB	*Binary Large Object*
CSS3	*Cascading Style Sheets*, *version* 3
DLL	*Dynamic Link Library*
GAE	*Google App Engine*
GUI	*Graphical User Interface toolkit*
HTML 5	*Hypertext Markup Language, version* 5
HTTP	*Hypertext Transfer Protocol*
IaaS	*Infrastructure as a Service*
IBM	*International Business Machines*
IBM-PC	*IBM Personal Computer*
IDE	*Integrated Development Environment*
JNI	*Java Native Interface*
JSON	*JavaScript Object Notation*
LDAP	*Lightweight Directory Access Protocol*
Mac	*Macintosh Operating System*
MCW	*Managed Callable Wrappers*
MooTools	*My Object-Oriented Tools*
MS-DOS	*MicroSoft Disk Operating System*
MVC	*Model-view-controller*
PaaS	*Platform as a Service*
PAYG	*Pay-as-you-go*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*

PIM	*Personal Information Management*
POPL	*Psion Organizer Programming Language*
REST	*Representational State Transfer*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
TCO	*Total Cost Ownership*
TI	Tecnologias de Informação
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WA	*Windows Azure*
XML	*Extensible Markup Language*
XUI	*XML User Interface*

Apresentam-se as convenções tipográficas empregues neste documento:

- Aplica-se no texto normal a fonte *Arial*, o tamanho 12 e o espaçamento entre linhas de 1,5 cm.

  Exemplo: Texto normal;

- Aplicam-se parêntesis curvo nas referências bibliográficas.

  Exemplo: (1);

- Aplica-se texto em itálico, delimitado por aspas, nas frases ou expressões ilustrativas.

  Exemplo: "*Frase ilustrativa*";

- Aplica-se texto em itálico nas designações em inglês.

  Exemplo: *Worklight*;

- Aplica-se texto em negrito para definir acrónimos e siglas.

  Exemplo: *International Business Machines* (**IBM**).

1 PRÓLOGO

As modernas Tecnologias de Informação (**TI**) que suportam a gestão do negócio das empresas, designadamente operadoras das redes de transportes públicos coletivos de passageiros, convergem para a adoção de arquiteturas computacionais elásticas, adaptáveis às incertezas do mercado, que sejam facilitadoras reais do controlo de qualidade, da interoperabilidade e naturalmente da gestão dos custos inerentes ao seu funcionamento (1).

1.1 Motivação

No âmbito da bilhética de transportes, a dificuldade em minimizar os custos e a complexidade dos dispositivos, são dois fatores imprescindíveis para a otimização de recursos e gestão dos serviços oferecidos aos utentes. A heterogeneidade dos dispositivos (e.g. *hardware*) dificulta o aparecimento de serviços mais versáteis adaptados às necessidades emergentes do mercado.

1.2 Objetivo

O objetivo do presente trabalho consiste em propor uma arquitetura de *software* para a implementação de aplicações, no domínio da bilhética de transportes, para plataformas móveis (e.g. *Android* (2), *Windows Phone* (3), *Apple* iOS (4)) preparadas para interagirem com nuvens computacionais (e. g. *Windows Azure* (**WA)** (5), *Google App Engine* (**GAE**) (6) ou *Amazon Web Services* (**AWS**) (7)). Para avaliar a solução proposta, serão replicados dispositivos típicos da bilhética.

1.3 Bilhética de transportes

Os projetos tecnológicos no âmbito da bilhética de transportes, nas áreas de eficiência, acessibilidade e segurança nos transportes representam um setor imprescindível para completar a plataforma tecnológica dos transportes.

As operações como o carregamento e/ou validação de títulos de transporte em dispositivos móveis, interligados por um sistema central de informação multioperador, consistem na informatização de processos essenciais à prestação do serviço de transporte público (8).

1.4 Dispositivo móvel

A computação móvel refere-se ao uso de dispositivos móveis que oferecem a capacidade de manipular aplicações, dados e sistema (9).

Os dispositivos móveis têm em comum as seguintes caraterísticas:

a. **Portabilidade**: capacidade de ser transportável e é influenciada pelo tamanho e peso do dispositivo;
b. **Usabilidade**: depende de diversos fatores, tais como as caraterísticas do utilizador (e.g. conhecimento e capacidade), ambiente (e.g. temperatura) e do dispositivo (e.g. interface do utilizador);
c. **Funcionalidade**: implementada na forma de aplicações móveis;
d. **Conectividade**: função primária de um dispositivo móvel consiste em conectar pessoas ou sistemas, bem como, transmitir e receber dados/informações. A percentagem de conectividade dos dispositivos móveis face a um sistema externo é variável e determina a autonomia e atualização face à existência de novas versões.

Na Figura 1 são apresentados os fatores a considerar no desenvolvimento de aplicações para dispositivos móveis.

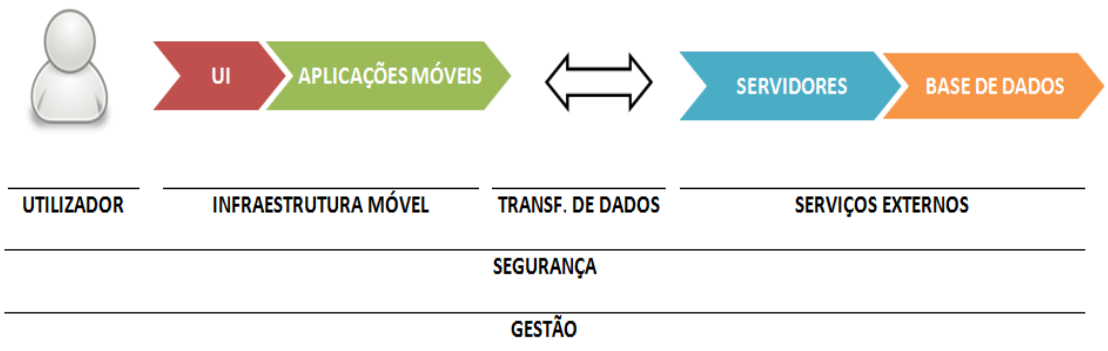

Figura 1 - Fatores no desenvolvimento de aplicações móveis

A interface do utilizador (UI) é constituída por um ecrã, teclado e um ponteiro (i.e. *pointing device*) e deve ser intuitiva e simples; a complexidade das aplicações móveis instaladas deve ser balanceada tendo em conta os recursos físicos do dispositivo (i.e. memória, processador e alimentação (i.e. bateria)) e a comunicação, se necessária, através de transferência de dados de e para serviços externos (e.g. servidores), para completar a lógica dos dispositivos. Os fatores de segurança e gestão do desenvolvimento de aplicações móveis são igualmente imprescindíveis para a integridade e eficiência do dispositivo móvel (10).

1.5 Computação em nuvem

O paradigma computação em nuvem disponibiliza infraestruturas teoricamente ilimitadas, virtuais, escaláveis, multiutilizador e com uma gestão automática de recursos.

As infraestruturas são constituídas por dispositivos de armazenamento e de processamento, com a capacidade de alojar aplicações heterogéneas, com adequados níveis de desempenho e segurança, cujo modo de pagamento é baseado no uso (i.e. *Pay-as-you-go* (**PAYG**)), ou seja, o utilizador paga pelos recursos utilizados durante o período contratado (11).

A possibilidade de minimização de custos a partir do modelo de pagamento referido é uma realidade pois torna desnecessário o pagamento de uma licença pelo seu uso integral do *software* pretendido, como um ambiente local de computação, com obrigatoriedades ao nível da manutenção da infraestrutura, nem mesmo a instalação e atualização de *software* nos computadores corporativos (12).

A virtualização inerente ao paradigma torna transparentes os recursos físicos do ambiente computacional, permitindo o acesso aos recursos da nuvem a partir de qualquer lugar, utilizando exclusivamente um dispositivo de acesso à Internet.

Assim, o paradigma introduz uma metodologia de negócio e de gestão dos recursos da infraestrutura, onde os seus utilizadores utilizam os recursos

físicos da infraestrutura (i.e. *hardware*), máquinas virtuais com uma base pré-instalada de *software* (i.e. sistema operativo) e aplicações completas, através de serviços disponibilizados na Internet (13).

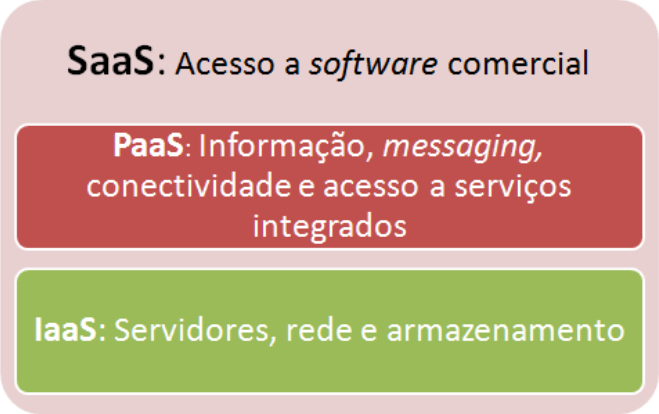

Figura 2 - Tipos de serviços da nuvem computacional

De acordo com a Figura 2, a nuvem computacional tem como serviço de mais baixo nível a infraestrutura como um serviço (i.e. *Infrastructure as a Service* (**IaaS**)), que disponibiliza os recursos físicos; posteriormente a plataforma como um serviço (i.e. *Platform as a Service* (**PaaS**)), que permite o desenvolvimento e a integração de aplicações e, por fim, aplicações como um serviço (i.e. *Software as a Service* (**SaaS**)), que disponibiliza através da Internet, aplicações desenvolvidas pelo fornecedor da nuvem e por terceiros (14).

1.6 Organização

Este documento está organizado em cinco capítulos.

No primeiro capítulo é concretizada a introdução, indicando a motivação, o objetivo, a indicação de aspetos gerais relativamente ao seguinte: bilhética de transportes, aplicação móvel e o paradigma computação em nuvem, e a estrutura do documento.

No segundo capítulo apresenta-se um estudo sobre a evolução dos dispositivos móveis, dos tipos de aplicações móveis, bem como uma análise das respetivas plataformas de desenvolvimento.

No terceiro capítulo é descrita a solução proposta, para a conceção de aplicações móveis com lógica balanceada, ou seja, o seu comportamento pode ser suportado parcialmente numa nuvem computacional.

No quarto capítulo é apresentado um caso de uso da arquitetura proposta para implementar dispositivos associados ao domínio da bilhética. Como exemplo prático, apresenta-se o demonstrador desenvolvido, no âmbito da plataforma móvel *Worklight* (15) e da nuvem computacional WA (16).

Por último, no quinto capítulo apresenta-se a conclusão destacando perspetivas de desenvolvimentos futuros.

No presente capítulo é apresentado o estado da arte dos dispositivos móveis e das respetivas plataformas de desenvolvimento existentes no mercado.

2.1 Dispositivos móveis

Os dispositivos móveis podem variar de tamanho, tecnologias suportadas e apresentação (i.e. *layout*), mas têm em comum um conjunto de características que os distinguem de sistemas *desktop*, nomeadamente (17):

a. **Tamanho reduzido**: os utilizadores finais desejam dispositivos pequenos pela sua mobilidade e peso reduzido.
b. **Memória limitada**: os dispositivos móveis possuem espaço de memória limitado e inferior a máquinas de secretária, o que condiciona o desenvolvimento de aplicações.
c. **Processamento limitado**: a capacidade de processamento é igualmente limitada e reduzida, face a máquinas de secretária.
d. **Bateria**: no desenvolvimento de aplicações para os dispositivos móveis tem de se garantir uma gestão eficiente da bateria, recorrendo a técnicas otimizadas, de modo a não comprometer a execução de outras aplicações.
e. **Reduzido tempo de arranque**: os dispositivos móveis iniciam-se em segundos devido à quantidade e simplicidade de serviços do dispositivo.

Sendo assim, o desenvolvimento de aplicações para dispositivos móveis requer ter em consideração as características e limitações anteriormente enumeradas.

2.1.1 Evolução

A evolução dos dispositivos móveis até à geração atual é ilustrada na Figura 3 (18).

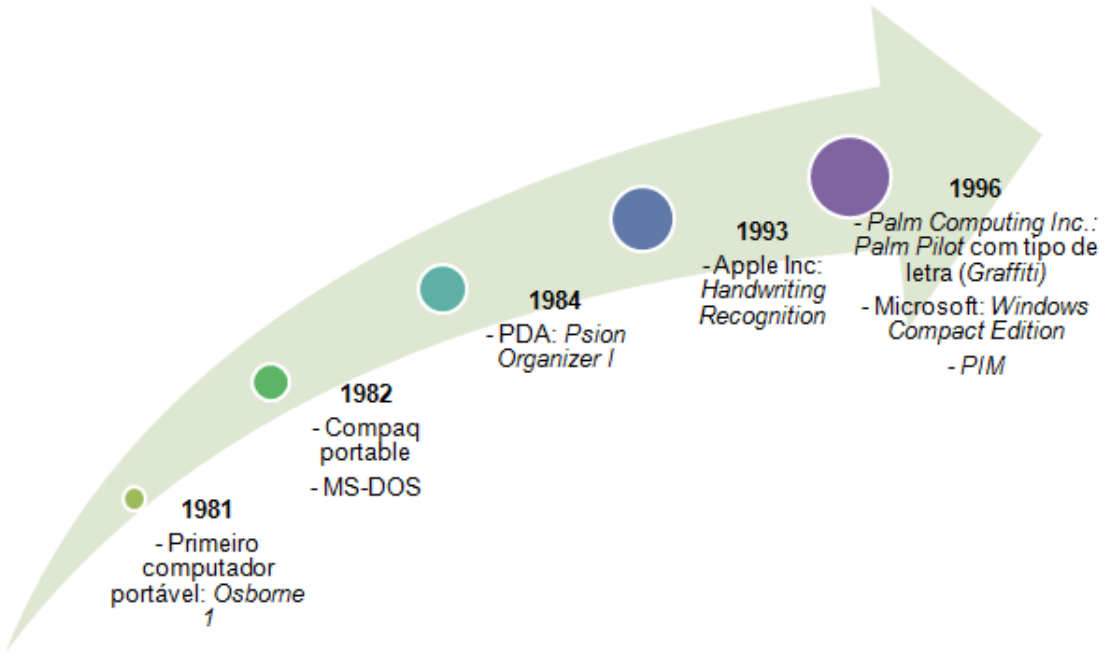

Figura 3 - Evolução dos dispositivos móveis

Em 1981, foi lançado o primeiro computador, designado por *Osborne 1* (Figura 4). O seu aspeto é bem diferente dos dispositivos móveis atuais pois apenas era capaz de mostrar 52 carateres no seu ecrã. Este computador destacou-se por incluir um conjunto de tecnologias como: *Beginner's All-purpose Symbolic Instruction Code* (**Basic**), *WordStar* e *SuperCalc* (19).

Figura 4 - O primeiro computador portátil - *Osborne 1*

No ano seguinte, a empresa Compaq lançou um dispositivo designado por *Compaq Portable*. Este tinha como benefício o facto de executar o sistema operativo *MicroSoft Disk Operating System* (**MS-DOS**) e por ser compatível com os programas desenvolvidos para a IBM *Personal Computer* (**IBM-PC**) (20).

Nos anos posteriores, surgiram novos dispositivos portáteis como, por exemplo, o *Radio Sharck TRS-80* (Figura 5), de uso doméstico e para pequenas empresas (21).

Figura 5 - Dispositivo portátil *Radio Sharck TRS*-80

Em 1984 surgiram os dispositivos denominados por *Personal Digital Assistant* (**PDA)**, pela empresa *Psion*. O primeiro tipo de PDA, designado por *Psion Organizer I,* integrava uma tecnologia de 8 bits, com base de dados, funções de pesquisa e um pacote com operações matemáticas. Este pacote foi desenvolvido de acordo com a linguagem *Psion Organizer Programming Language* (**POPL**) (22).

O lançamento do *Psion* II deu-se em meados da década de 80 e início dos anos 90, onde foram produzidas cerca de 500 000 unidades (23).

Em 1993, surgiu a terceira série destes dispositivos, com a capacidade de comunicação com um computador *Personal Computer* (**PC**). Com esta funcionalidade surgiu a possibilidade de realizar transferências, conversões e sincronização de dados entre dois ambientes distintos (24).

O sucesso da empresa *Psion* no mercado dos dispositivos portáteis despertou noutros fabricantes o interesse em expandir o mercado. Um exemplo foi a empresa *Apple Inc*. que lançou um dispositivo, denominado por *Newton MessagePad* (Figura 6)*,* com a tecnologia de reconhecimento de texto escrito à mão (*Handwriting Recognition*). Através desta funcionalidade, o utilizador podia interagir com o dispositivo através de uma caneta e, inclusive, permitia que o dispositivo interpretasse a escrita, convertendo-a em texto (25).

Figura 6 - *Apple Newton MessagePad*

Em 1996, a empresa *Palm Computing Inc*. desenvolveu o primeiro *Personal Digital Assistant* (**PDA**), designado por *Palm Pilot.* Este dispositivo incluía o seu próprio tipo de letra (i.e. *Graffiti*), o que facilitava o reconhecimento escrito pelo utilizador. Três anos depois, esta empresa dominou cerca de 70% do mercado de PDA, o que favoreceu o aumento do número aplicações (26).

No mesmo ano, a empresa *Microsoft* lançou o seu primeiro sistema operativo, para dispositivos móveis, designado por *Windows Compact Edition.* Este sistema teve evoluções, por exemplo, ao nível da interface gráfica, de modo a garantir uma sucessiva adaptação às caraterísticas dos dispositivos portáteis da época. Um exemplo dessa evolução foi uma versão designada por *Personal Information Management* (**PIM**). O PIM era uma aplicação com o objetivo de facilitar o registo e a gestão de informações pessoais (27).

2.1.2 Ciclo de vida

A proliferação de dispositivos de consumo está a mudar as formas através das quais as empresas oferecem soluções de mobilidade para o mercado de trabalho. As empresas de TI são forçadas a criar programas de mobilidade (e.g. *e-mail* corporativo) e produtos de consumo, tais como: *iPhone*, *iPad* e dispositivos *Android* (28).

As aplicações móveis têm tido um crescimento exponencial e têm influenciado, desde o seu aparecimento, diversas indústrias. Apesar de existirem aspetos comuns entre os desenvolvimentos para plataformas móveis e para máquinas de secretária, o desenvolvimento para dispositivos móveis apresenta aspetos únicos (29).

Devido aos reduzidos recursos computacionais de processamento e armazenamento dos dispositivos móveis, as aplicações necessitam de estar ligadas à sua organização, de modo, a manipularem dados e a manterem os serviços que disponibilizam (30).

Os fatores portabilidade, o custo total de propriedade *Total Cost Ownership* (**TCO**), a riqueza da camada de apresentação e dos serviços variam consoante o tipo de aplicação.

O ciclo de vida de aplicações móveis é constituído pelas fases de negócio, desenvolvimento e de produção. É fundamental que o ciclo seja conciso e eficaz, de modo a proporcionar a rápida e frequente saída de requisitos para as aplicações móveis. A Figura 7 apresenta o ciclo mencionado (31) (32).

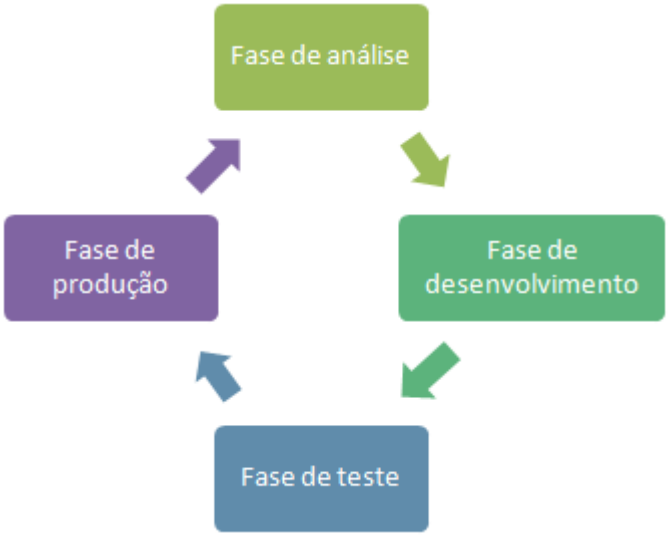

Figura 7 - Ciclo de vida do desenvolvimento de aplicações móveis

Na fase de análise são definidos os requisitos de negócio, de modo a atender as necessidades dos clientes, que esperam maior qualidade de experiência, face às aplicações tradicionais.

Na fase de desenvolvimento, a equipa de responsáveis inicia o seu trabalho na fase de desenho da arquitetura, de modo a cumprir o processo de negócio.

A fase seguinte consiste no teste onde é verificado se o protótipo em desenvolvimento cumpre os requisitos definidos na fase de análise, garantindo a qualidade das configurações face às características dos dispositivos alvo. Por

fim, se o protótipo passar nos testes realizados é colocado em produção (33) (34).

2.2 Comparação entre tipos de aplicações móveis

O processo de escolha de uma abordagem de desenvolvimento de uma aplicação móvel nativa, direcionada para a *web* (i.e. *World Wide Web*) ou híbrida, requer a ponderação nos seguintes fatores: orçamento, cronograma do projeto alvo, audiência e funcionalidades. As abordagens referidas são apresentadas na Figura 8 (35) (36).

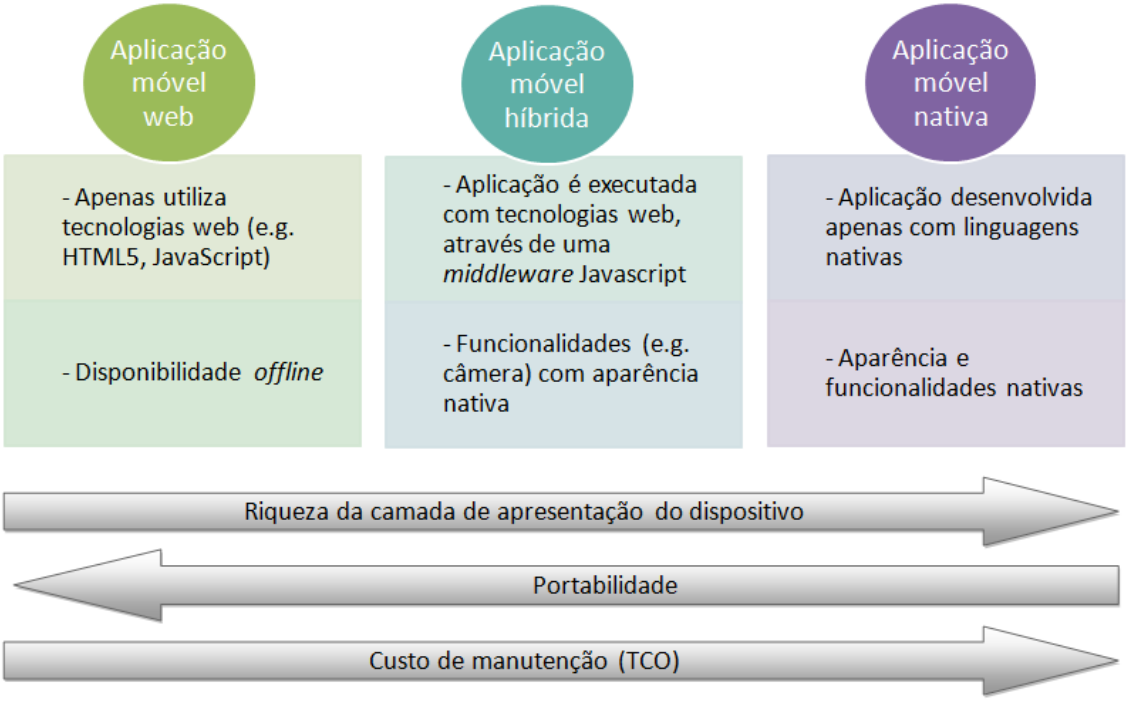

Figura 8 - Abordagens no desenvolvimento de uma aplicação móvel

A opção de uma aplicação nativa supera em desempenho de acesso e na riqueza da camada de apresentação (i.e. *User Interface* (**UI**)), mas tem como desvantagens o custo, a dificuldade em realizar atualizações remotas e a portabilidade entre plataformas.

A abordagem *web* é considerada mais simples, de custo mais reduzido e com facilidades ao nível da atualização e da portabilidade da aplicação, mas está atualmente limitada ao nível das funcionalidades disponíveis, pois algumas apenas estão disponíveis na opção nativa.

A preferência por uma aplicação híbrida oferece um meio-termo que, em muitas situações, é o melhor dos dois mundos, especialmente se a aplicação tem como alvo mais do que uma plataforma.

2.2.1 Aplicação nativa

As aplicações nativas são constituídas por ficheiros executáveis que são diretamente descarregados no dispositivo e armazenadas localmente. O processo de instalação pode ser iniciado pelo utilizador ou, em alguns casos, pelo departamento TI da empresa. O processo mais comum para o descarregamento de uma aplicação móvel nativa consiste em aceder a um repositório de aplicações (e.g. *Apple App Store* (37) , *Google Play* (38), *BlackBerry App World* (39)).

Após a instalação da aplicação no dispositivo, o utilizador pode executá-la, como qualquer outro serviço disponível. Depois do arranque, a interface da aplicação nativa interage diretamente com o sistema operativo, sem intermediários. A aplicação é livre de aceder a qualquer uma das interfaces *Application Programming Interface* (**API**), disponibilizadas pelo fornecedor, e em muitos casos, têm caraterísticas únicas e funções que são típicas do dispositivo em uso (40).

Para desenvolver uma aplicação nativa, os programadores escrevem o código fonte e adicionam outros recursos, como imagens e sons. Posteriormente, recorrendo a ferramentas do fornecedor (e.g. *Software Development Kit* (**SDK**)), é gerado um ficheiro, que empacota todas partes desenvolvidas.

O ficheiro produzido está comprometido com o formato específico do SDK utilizado. Este facto é uma das desvantagens associadas ao desenvolvimento de aplicações nativas. O código gerado não pode ser reutilizado por outra plataforma. Deduz-se, portanto, que o desenvolvimento e a manutenção de aplicações nativas para múltiplas plataformas implicam compromissos de longa duração e de elevados custos.

Em oposição, um dos benefícios inerentes ao desenvolvimento de aplicações nativas depreende-se do facto de se poder utilizar API específicas (e.g.

Graphical User Interface toolkit (**GUI**)), com caraterísticas e funções específicas, o que, normalmente resulta numa interface gráfica mais agradável e fácil de usar.

Assim, torna-se importante salientar que apesar das API específicas adicionarem demasiada complexidade e custo no desenvolvimento de aplicações nativas, estes componentes são o único meio para criar aplicações que fazem pleno uso de todas as funcionalidades que os dispositivos oferecem.

2.2.2 Aplicação *web*

Atualmente, os recentes dispositivos móveis destacam-se das suas versões anteriores pelo facto de integrarem navegadores (i.e. *browsers*) com tecnologias recentes, tais como: *Hypertext Markup Language, version* 5 (**HTML5**) (41) , *Cascading Style Sheets, version* 3 (**CSS3**) (42) e *JavaScript* (43). Com estas tecnologias, sinaliza-se a transição de páginas simples para o desenvolvimento de aplicações ricas e poderosas (e.g. serviços de geolocalização, disponibilidade *offline*).

As aplicações para dispositivos móveis são uma tendência de evolução promissora. Para capitalizar esta tendência e ajudar os projetistas a criarem as interfaces do lado do cliente, foram criados um número crescente de ferramentas baseadas na tecnologia, tais como: *Dojo Mobile* (44), *Sencha Touch* (45) e *jQuery Mobile* (46). Estas ferramentas geram interfaces gráficas comparáveis, em aparência, como as das aplicações nativas.

Uma das vantagens eminentes do desenvolvimento de aplicações *web* é o suporte de multiplataformas e o baixo custo do seu desenvolvimento. O código destas aplicações é escrito em linguagens de tecnologias *web*, o que proporciona que uma mesma aplicação possa ser integrada em diferentes dispositivos com sistemas operativos distintos.

Por outro lado, ao contrário das aplicações nativas que são executáveis independentes com acesso direto ao sistema operativo, as aplicações *web* são executadas no contexto de um navegador. Este, por sua vez, é igualmente uma aplicação nativa, que acede diretamente ao sistema operativo do dispositivo.

Assim, por existir um número reduzido de API que disponibilizam a sua interface para aplicações *web*, este facto traduz-se numa das desvantagens no desenvolvimento de aplicações *web*.

2.2.3 Aplicação híbrida

Uma aplicação híbrida combina o desenvolvimento de código nativo com outras tecnologias. Com esta opção, os programadores implementam parte das aplicações, recorrendo a plataformas com tecnologias *web* e mantêm o acesso direto com as API nativas.

A parte nativa da aplicação utiliza a interface do sistema operativo para criar um mecanismo de estruturação de elementos HTML, que serve como uma camada intermédia entre o navegador e as API do sistema operativo do dispositivo. Esta camada permite que a aplicação híbrida tire o máximo proveito de todas as características que os dispositivos modernos têm para oferecer aos seus utilizadores.

Os programadores podem optar por desenvolver a sua própria camada intermédia ou recorrer a soluções já existentes (e.g. *PhoneGap*), que disponibilizam uma interface uniforme, em *JavaScript*, que permite selecionar as capacidades do dispositivo, garantindo a interoperabilidade entre sistemas operativos.

A parte nativa da aplicação híbrida pode ser desenvolvida de forma independente. Contudo, existem atualmente no mercado soluções nativas específicas do dispositivo alvo, o que permite ao programador personalizar a aplicação nativa, com as necessidades específicas do utilizador final.

Para o desenvolvimento da parte *web*, pode-se optar por uma das seguintes soluções: resumir-se a uma página que reside num servidor ou num conjunto de ficheiros (e.g. HTML, *JavaScript*, CSS), integrados no código da aplicação e, portanto, armazenados localmente no dispositivo.

A primeira opção implica que caso ocorram atualizações na aplicação, estas sejam realizadas no servidor. Contudo, esta opção elimina a possibilidade de

funcionamento *offline,* pois o conteúdo não está acessível, quando o dispositivo não está ligado à rede da Internet.

Por outro lado, integrar os ficheiros na aplicação permite melhores níveis de desempenho e acessibilidade mas não possibilita atualizações remotas. O melhor das duas opções consiste na combinação de ambas, ou seja, manter os ficheiros num servidor mas garantindo que os mesmos existem localmente no dispositivo (i.e. *cache*) (47).

2.2.4 Comparação dos tipos de aplicações móveis

Nesta secção apresenta-se uma comparação entre os três tipos de aplicações móveis analisadas, nativa, *web* e híbrida. Pretende-se comparar no que diz respeito à existência de dependência do fornecedor, à possibilidade de reutilização de código, se existe algum compromisso, a longa duração, na adoção de cada um dos tipos, o custo inerente ao desenvolvimento multiplataforma e a comparação no acesso a recursos do dispositivo nos três tipos de aplicações (Tabela 1).

	Aplicação nativa	**Aplicação *web***	**Aplicação híbrida**
Dependência do fornecedor	Sim	Não	*
Reutilização do código	Não	Sim	*
Compromisso a longa duração	Sim	Não	*
Multiplataforma	Não	Sim	Sim
Custo de desenvolvimento multiplataforma	Alto	Baixo	*
Acesso a recursos do dispositivo	Mais do que nos restantes tipos de aplicações.	Menos do que nos restantes tipos de aplicações.	Mais do que no tipo *web* e menos do que no tipo nativo.

Tabela 1 - Comparação entre os tipos de aplicações móveis analisadas

* Depende do grau de acoplamento do código híbrido face ao SDK do fornecedor.

2.3 Plataformas de desenvolvimento de aplicações móveis

Na presente secção realiza-se uma análise de ferramentas de desenvolvimento para aplicações móveis multiplataforma, existentes no mercado, por designadamente: *PhoneGap* (48)*, Appcelerator* (49)*, Xamarim* (50) e *Worklight* (51).

2.3.1 Plataforma *Phonegap*

A empresa *Nitobi,* incorporada na *Adobe*, concebeu uma plataforma, de código aberto, para o desenvolvimento de aplicações móveis, denominada por *PhoneGap* (52) (53).

A plataforma *PhoneGap* é considerada uma ponte entre os mundos *JavaScript* e nativo. Esta é sustentada por uma arquitetura (Figura 9) que fornece um mecanismo de mapeamento de funções *JavaScript* em código nativo, ou seja, permite adicionar código nativo a aplicações *PhoneGap* e expô-lo através de código *JavaScript.* Ambos estão integrados na respetiva plataforma (54).

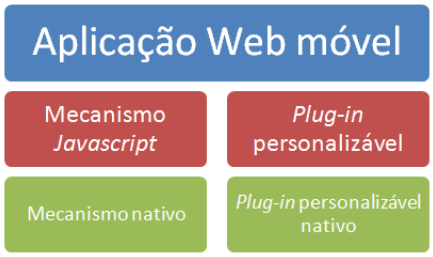

Figura 9 - Arquitetura da plataforma *PhoneGap*

Contudo, para não tornar as aplicações móveis demasiado pesadas deverá ser ponderado o modo de implementação das mesmas, se em código nativo se em *JavaScript.* Esta última não deverá ser utilizada em operações complexas ou de *background,* por motivos de desempenho. Nestas situações, as operações deverão ser implementadas em código nativo *PhoneGap.*

Um exemplo da ponderação referida consiste no descarregamento de um ficheiro constituído por múltiplas partes. Esta operação deverá ser realizada em paralelo e no final deverá ser verificado o respetivo *checksum.* Nesta situação

deverá optar-se por código nativo Java para o sistema operativo *Android* (55) ou *Objective-C* para *iPhone* (56).

A plataforma inclui um componente designado por *PhoneGap Build*. Trata-se de um serviço *online,* que empacota aplicações móveis desenvolvidas em HTML, CSS e *JavaScript,* de modo a serem integradas em dispositivos com sistemas operativos, tais como: *Apple* iOS (57), *Google Android* (58), *Palm* (59), *Symbian* (60) e *BlackBerry* OS (61).

As aplicações desenvolvidas na plataforma *PhoneGap* são híbridas, ou seja, nativas e *web*, com a possibilidade de utilizar serviços externos, tais como: agendas, câmaras e notificações (e.g. som, vibração) (62).

Para preencher as necessidades das linguagens nativas, a plataforma *PhoneGap* utiliza padrões *web*. A interface é baseada em elementos HTML5, formatação em CSS3 e a lógica através da linguagem *JavaScript,* onde é incorporada a comunicação com a API *PhoneGap* (63). A linguagem *JavaScript* pode ser combinada com outros *scripts,* tais como: *jQuery* (64), *My Object-Oriented Tools* (**MooTools**) (65) e XML *User Interface* (**XUI**) (66).

A plataforma disponibiliza padrões para o armazenamento de dados, nomeadamente: SQLite (67) e *LocalStorage* (68). O primeiro trata-se de uma biblioteca em linguagem C que tem embutido um banco de dados *Structured Query Language* (**SQL**). O segundo é um modelo de armazenamento local, que permite guardar os dados no lado do cliente, como se fossem *cookies*. Uma das diferenças consiste no facto do *LocalStorage* não ter associado uma data de expiração, como acontece no modelo dos *cookies* (69).

2.3.2 Plataforma *Titanium*

A plataforma móvel *Appcelerator Titanium* (**AT**) é uma plataforma aberta, para o desenvolvimento de aplicações nativas para dispositivos de diferentes tecnologias (iOS*, Android, Windows* e *BlackBerry*) e inclusive aplicações híbridas (70) (71).

Trata-se de uma plataforma de desenvolvimento móvel modular, constituída por *JavaScript e* código nativo (*Java*, *Objective-C* e C++). A arquitetura fornece um mecanismo de mapeamento entre o *runtime JavaScript* e uma *Application Programming Interface* (**API**) disponibilizado para o desenvolvimento móvel nativo (Figura 10) (72).

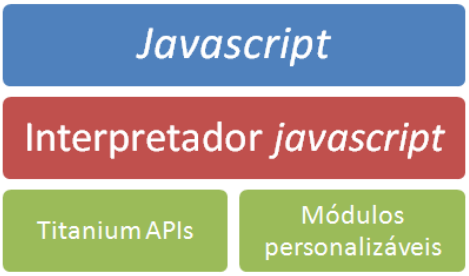

Figura 10 - Arquitetura da plataforma *Titanium*

Cada aplicação desenvolvida a partir da plataforma AT é organizada numa arquitetura de camadas, nomeadamente:

a. **Código *JavaScript* desenvolvido:** em tempo de compilação, o código *JavaScript* é codificado e armazenado em ficheiros Java ou *Objective-C.*
b. **Interpretador *JavaScript Titanium:*** como exemplos existem o *Android V8* (73) ou *JavaScriptCore* para iOS (74).
c. **API Titanium:** esta API é específica para plataformas desenvolvidas em Java, *Objective-C* ou C++.
d. **Módulos nativos personalizáveis:** são disponibilizados, pela plataforma AT, diversos módulos de código aberto (49).

Em tempo de execução, o SDK *Titanium,* incluído nas aplicações desenvolvidas, cria um contexto de execução em código nativo. Esta execução é posteriormente avaliada no código *JavaScript* das aplicações. Quando o código *JavaScript* é executado, é criado um *proxy* que acede ao código nativo através das APIs disponibilizadas. Estes *proxies* são responsáveis pelo mapeamento entre os dois ambientes, código nativo e *JavaScript.* Por exemplo, se a cor de fundo de um objeto da interface do utilizador for alterada, esta operação é realizada em *JavaScript* e, seguidamente, o respetivo *proxy* atualiza a propriedade correta do objeto na camada nativa (72).

A plataforma AT inclui um SDK, de fonte aberta, com mais de 5.000 dispositivos e API de acesso a sistemas operativos móveis. A plataforma disponibiliza igualmente um ambiente de desenvolvimento (i.e. *Integrated Development Environment* (**IDE**)) Java e uma *framework,* baseada num modelo de desenvolvimento de *software, Model-View-Controller* (**MVC**), designado por *Appcelerator Alloy* (**AA**) (75).

AT é uma plataforma que permite aos programadores cumprirem as fases constituintes do desenvolvimento de aplicações (i.e. construir, testar, *package* e publicar), recorrendo a bibliotecas *JavaScript* (76).

A plataforma disponibiliza serviços direcionados para as nuvens computacionais, designado por *Appcelerator Cloud Services* (**ACS**) (77). Este serviço permite a integração de outros serviços públicos, de modo a interligar serviços de terceiros, para expandir as funcionalidades e utilidades, segundo uma metodologia de pagamento baseada no uso (PAYG) (78).

Por se tratar de serviços alojados na nuvem, o serviço ACS garante a elasticidade de recursos, a segurança (i.e. integridade e privacidade dos dados), a disponibilidade dos serviços móveis e, ao contrário das aplicações tradicionais, retira ao cliente o foco na complexidade de servidor (e.g. gestão de servidores e rede), permitindo que se concentre no seu negócio (79) (80).

2.3.3 Plataforma *Xamarin*

A plataforma *Xamarin* permite o desenvolvimento de aplicações móveis interoperáveis entre sistemas operativos móveis, como: *Apple* iOS, *Google Android, Macintosh Operating System* (**Mac OS**) e *Windows* (81).

Esta plataforma possibilita o desenvolvimento de aplicações nativas, garantindo o acesso a API nativas, específicas dos dispositivos alvo (e.g. *Apple* iOS, *Android*). A utilização da plataforma .NET permite inclusive a utilização de bibliotecas .NET e, consequentemente, a reutilização de código entre diferentes plataformas. Esta caraterística é possível porque a plataforma garante a separação entre a UI dos diferentes sistemas operativos móveis e as restantes camadas de código. A componente nativa é feita através da interação

das API específicas de cada sistema operativo, juntamente com a componente *web*, que permite a partilha da lógica, acessos de dados e comunicações de rede. O objetivo é garantir a reutilização do código em vários sistemas operativos (i.e. "*write-once, run everywhere*") (82).

As aplicações *Xamarin Android* são executadas num ambiente de execução, *Mono,* em paralelo com uma máquina virtual designada por *Dalvik.* Ambos os ambientes foram desenvolvidos recorrendo à linguagem C, correm em cima de um *kernel Linux* e disponibilizam APIs (e.g. bibliotecas .Net) que dão acesso ao sistema subjacente (83) (Figura 11).

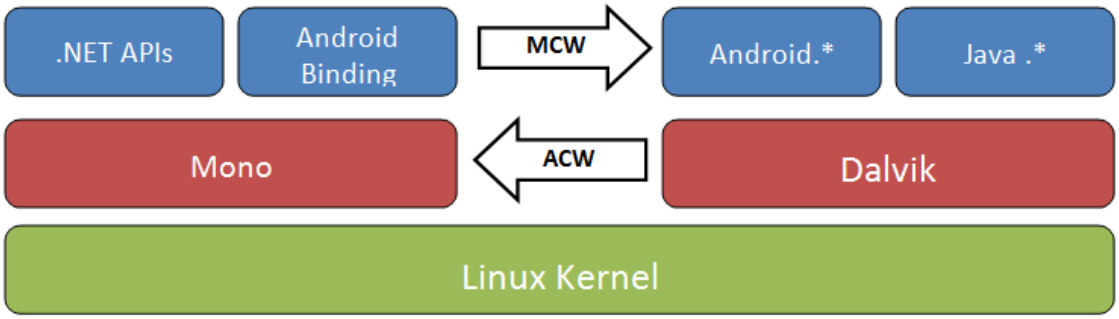

Figura 11 - Arquitetura da plataforma *Xamarin*

O mapeamento entre os dois ambientes de execução, *Mono* e *Dalvik,* é feito por dois componentes designados por *Android Callable Wrappers* (**ACW**) e *Managed Callable Wrappers* (**MCW**). O primeiro é uma *Java Native Interface* (**JNI**) *bridge* e é utilizado pelo *runtime Android* para a invocação de código *managed* (84). O segundo é igualmente uma JNI *bridge* e permite que no código *managed* seja invocado código *Android,* com suporte para sobrepor (i.e. *override*) métodos virtuais e a implementação de interfaces Java.

Quanto a ferramentas de desenvolvimento, a plataforma *Xamarin* permite que sejam utilizados o *software Microsoft Visual Studio* ou uma ferramenta fornecida pela plataforma, designada por *Xamarin Studio* que integra um IDE específico para o desenvolvimento de aplicações móveis, no âmbito do sistema operativo Mac OS e *Windows* (85).

No âmbito da *cloud,* a plataforma *Xamarin* disponibiliza uma ferramenta, designada por *Xamarin Test Cloud,* que permite o teste das aplicações móveis,

independentemente do seu sistema operativo alvo, desenvolvidas no contexto em vários dispositivos móveis (86).

2.3.4 Plataforma *Worklight*

A empresa IBM associou-se à plataforma *Worklight*, originando a plataforma IBM *Worklight*. Esta plataforma fornece, de um modo aberto e avançado, metodologias para o desenvolvimento de aplicações móveis para dispositivos, tais como: *smartphones* e *tablets* (87).

A plataforma executa e gere aplicações nativas ou híbridas, recorrendo a tecnologias padrão (e.g. HTML5) disponibilizando um ambiente otimizado e apoiado em mecanismos de segurança, para desenvolver, gerir e realizar a instalação de aplicações (88).

A plataforma IBM *Worklight* já disponibiliza padrões para aplicações móveis. Os clientes da IBM desenvolvem as suas aplicações suscetíveis de serem executadas em dispositivos móveis de diferentes tecnologias (e.g. *Android*, *Windows Phone, Apple* iOS*, BlackBerry*). Esta plataforma permite a ligação de aplicações *backend* a sistemas de informação, possibilitando que sejam cumpridos os requisitos impostos pelo mercado (15).

A plataforma permite o desenvolvimento de aplicações, híbridas ou nativas, para o suporte de dispositivos recentes, utilizando código nativo e de tecnologias *web* padrão (e.g. *jQuery Mobile*, *Sencha Touch* e *Dojo Mobile*). A combinação de implementações facilita ao programador ponderar e balancear a eficiência e a riqueza da camada de apresentação.

A plataforma IBM *Worklight* é constituída por quatro componentes, nomeadamente*: Studio*, *Server, Device Runtime* e *Console.* A Figura 12 apresenta uma visão geral dos componentes (87) (89) (90).

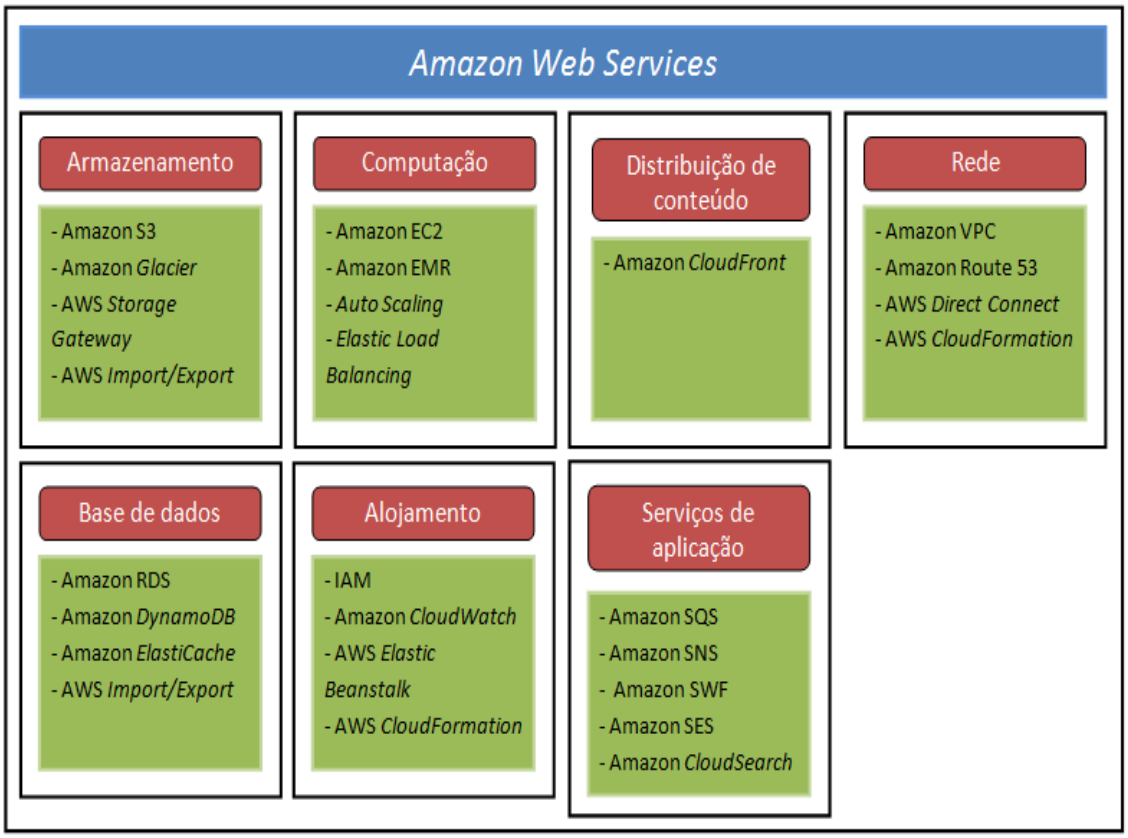

Figura 12 - Visão geral dos componentes de plataforma IBM *Worklight*

O componente IBM *Worklight Studio* disponibiliza recursos para o desenvolvimento de *software* (i.e. *Integrated Development Environment* (**IDE**)). Este componente está direcionado para o desenvolvimento de aplicações para dispositivos móveis, cujo código gerado é portável para outras plataformas. Os benefícios associados à utilização desta plataforma são, designadamente:

a. Desenvolvimento de aplicações nativas e híbridas, utilizando tecnologias *web* (e.g. HTML5).
b. Partilha de código enquanto se define comportamento personalizado e diretrizes de estilo, que correspondem ao ambiente de destino.
c. Acesso a API, utilizando código nativo ou linguagens *web* padrão.
d. Utilização de linguagens nativas e *web*, na mesma aplicação, de modo a balancear a eficiência e riqueza da interface gráfica.
e. Utilização de outras bibliotecas e *frameworks,* tais como: *jQuery Mobile, Sencha Touch* e *Dojo Mobile*.

O componente IBM *Worklight Server* é um servidor Java que é acedido através de um portal escalável que interliga aplicações/serviços externos à

infraestrutura da empresa. Este servidor tem recursos de segurança, permite extração, manipulação, autenticação e atualização dos dados da Internet e de aplicações híbridas.

O servidor foi projetado de modo a ser integrado no ambiente empresarial, para escalar os recursos existentes. O paradigma de integração é baseado em adaptadores (i.e. *adapters*), ou seja, componentes de *software* de servidor, responsáveis por interligar o sistema *backend* da empresa e os serviços da nuvem.

Os adaptadores podem ser utilizados para receber e/ou atualizar dados, enviados por outras fontes de informação, bem como, para disponibilizar mecanismos transacionais e de invocação de outros serviços e/ou aplicações.

Como principais benefícios de utilização do servidor, destacam-se os seguintes:

a. Configura, testa e implementa ficheiros *Extensible Markup Language* (**XML**) (91), de modo a ligar-se aos diversos sistemas *backend,* utilizando ferramentas padrão.
b. Disponibiliza aplicações num repositório privado, de alta disponibilidade, que cumpre o protocolo *Lightweight Directory Access Protocol* (**LDAP**) (92).
c. Integração através do componente IBM *Endpoint Manager* para dispositivos móveis, de modo a automatizar a implementação das aplicações na plataforma IBM *Worklight*.
d. Mecanismos de segurança garantidos pelo IBM *Worklight Server*.
e. Atualizações diretas para aplicações híbridas e da Internet, sem a necessidade de atualizar inúmeros contentores.
f. Converte automaticamente dados hierárquicos para um formato leve, que permite formatar o intercâmbio de dados, designado por *JavaScript Object Notation* (**JSON**) (93).
g. Define *mashups* (i.e. página da Internet personalizado ou uma aplicação *web*, com conteúdo proveniente de mais de uma fonte, para originar um serviço mais completo (94)), complexos de múltiplas fontes de dados para reduzir o tráfego geral.

h. Integra novos componentes com os mecanismos de segurança (e.g. autenticação) do cliente.

O componente IBM *Worklight Device Runtime Components* está desenvolvido em código cliente, porém inclui funcionalidades de servidor. Estas funcionalidades são essencialmente bibliotecas que são integradas localmente e que complementam o componente IBM *Worklight Server,* disponibilizando uma interface, predefinida para aplicações, que permite o acesso a funcionalidades nativas dos dispositivos. O componente IBM *Worklight Device Runtime Components* utiliza uma *framework* de desenvolvimento, de modo a disponibilizar um formato uniforme a ser utilizado por tecnologias *web* padrão (i.e. HTML5, CSS3, *JavaScript*) e as funções nativas que diferentes plataformas móveis disponibilizam (95).

As principais caraterísticas são, designadamente (96):

a. Integração de dados móveis.
b. Mecanismos de segurança (e.g. encriptação, autenticação *offline* e desativação remota de aplicações).
c. Suporte multiplataforma: abstrações de UI e compatibilidade de ferramentas HTML5.
d. Camada de desenvolvimento de aplicações híbridas, acesso a API de dispositivos e mecanismos de notificação (i.e. *push notification registration*).
e. Base de dados para grande quantidade de dados, com mecanismos de encriptação e de sincronização.
f. Relatórios personalizados e análises, baseados em eventos.
g. Atualização direta de recursos de aplicações *web* e de *cache* HTML5.

O componente IBM *Worklight Console* é uma aplicação *web*, dedicada para monitorização e administração do componente *Worklight Server* e as suas aplicações, adaptadores e mecanismos de notificação (i.e. *push notifications*).

O principal objetivo deste componente consiste no controlo contínuo e gestão da organização móvel, para obter estatísticas para os clientes (95).

Como principais caraterísticas evidenciam-se, nomeadamente:

a. Monitorizar as aplicações, adaptadores e mecanismos de notificação, centralizando a informação adquirida a partir de uma consola *web*.
b. Atribuir identificadores para garantir o aprovisionamento seguro de aplicações.
c. Desativar remotamente aplicações baseadas na pré-configuração de regras da versão da aplicação e do tipo de dispositivo.
d. Personalizar as mensagens enviadas para os clientes no arranque das aplicações.
e. Obter estatísticas a partir de aplicações em execução.
f. Configurar regras para específicos eventos associados a aplicações.
g. Exportar dados de relatórios para serem analisados pelos sistemas *Business Intelligence* (**BI**).
h. Simular diferentes dipositivos, através do navegador e simulador móvel que tem incluído.

Resumidamente, a plataforma IBM *Worklight* destaca-se por se tratar de uma plataforma aberta e otimizada permitindo escolher a tecnologia mais apropriada para o negócio, nativa, *web* ou híbrida, assim como as bibliotecas padrão. O ambiente de desenvolvimento não gera código, nem utiliza componentes que transformam o código, mas sim permite ao seu utilizador ter total controlo do código. Outro benefício consiste em disponibilizar um ambiente de execução que permite a interação com outros dados e serviços, através de componentes designados por adaptadores (97).

2.3.5 Comparação das plataformas móveis

Nesta secção apresenta-se uma comparação entre as plataformas de desenvolvimento de aplicações móveis analisadas, nomeadamente, *PhoneGap,* AT*, Xamarin* e *IBM Worklight*.

A Tabela 2 mostra a comparação das plataformas móveis referidas, no que diz respeito ao tipo de plataforma, ferramenta de desenvolvimento disponibilizada, as tecnologias utilizadas para o desenvolvimento das aplicações móveis, os sistemas operativos onde poderão ser instaladas as aplicações, o tipo de

aplicação gerada, os tipos de armazenamentos disponibilizados e o ambiente disponibilizado direcionado para a nuvem computacional.

				
Designação	*PhoneGap*	*Appcelerator Titanium*	Xamarin	IBM *Worklight*
Plataforma	Aberta e paga	Aberta e paga	Aberta e paga	Aberta e paga
Ferramenta de desenvolvimento	*PhoneGap Build*	*Appcelerator Alloy, IDE Eclipse*	*Xamarin Studio* e IDE *Microsoft Visual Studio*	IBM *Worklight Studio*
Tecnologias	HTML5, CSS3 e *JavaScript*	HTML, CSS3 e *JavaScript*	HTML, CSS3 e *JavaScript*	HTML5, CSS3 e *JavaScript*
Sistemas operativos	*Apple iOS, Google Android, Palm, Symbian, BlackBerry OS, Web OS, etc.*	*Apple iOS, Google Android, Windows e BlackBerry*	*Apple iOS, Google Android, Mac OS* e *Windows*	*Google Android, Windows Phone, Apple iOS, BlackBerry*
Aplicação	Híbrida	Nativa	Nativa e híbrida	Nativa, *web* e híbrida
Armazenamento de dados	*SQLite* e *Local Storage*	*SQLite*	*SQLite, Adapters, ContentProviders, Files*	SQLite, *database engine,*
Cloud	*PhoneGap Build Cloud*	*Appcelerator Cloud Services,* PAYG	*Xamarin Test Cloud*	*Adapters*

Tabela 2 - Comparação entre as plataformas móveis analisadas

A decisão da escolha da plataforma a utilizar para o desenvolvimento de aplicações móveis deverá ter em conta determinados parâmetros, tais como: o tipo de dispositivos; a escalabilidade de diferentes orientações e resoluções (e.g. UI, aplicações multi-plataforma), de modo a abranger um maior conjunto de dispositivos móveis; o nível de segurança que se pretende ter na aplicação; o tipo de abordagem de desenvolvimento permitido (nativo, web ou híbrido),

tendo presente os benefícios e limitações associadas; a disponibilização de uma ferramenta de colaboração, com um mecanismo de construção centralizada; entre outras (98).

No presente capítulo propõe-se uma arquitetura de *software* para a implementação de aplicações, no domínio da bilhética de transportes, ou seja, uma arquitetura transversal a várias plataformas móveis (e.g. *Android* (2), *Windows Phone* (3), *Apple* iOS (4)), preparada para interagir com nuvens computacionais (e.g. WA, GAE, AWS).

A proposta tem como objetivo facilitar o balanceamento de camadas aplicacionais entre o dispositivo móvel e uma nuvem computacional dependendo, por exemplo, das condições da rede e/ou dos recursos computacionais existentes no dispositivo móvel.

3.1 Contexto de utilização

Nesta secção é descrito o contexto de utilização do projeto assumindo, a título de exemplo, que a aplicação móvel possui três camadas. A Figura 13 ilustra o dispositivo móvel, onde residem duas camadas (i.e. *Layer 1* e *Layer 2*) da sua *Business Layer* (Figura 15), e a nuvem onde reside a terceira camada (i.e. *Layer 3*) (98) (99).

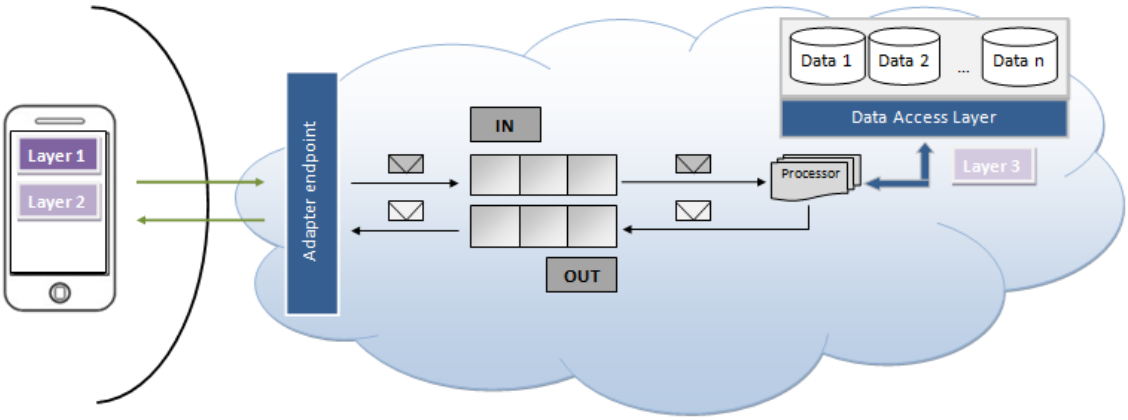

Figura 13 - Arquitetura proposta

O dispositivo envia o seu primeiro pedido para o adaptador (i.e. *Adapter endpoint*) e, este por sua vez reencaminha-o para a nuvem computacional, através de uma mensagem, pela fila de mensagens de entrada (i.e. *IN*). Na

nuvem, a mensagem recebida é interpretada por componentes, designados por processadores (i.e. *Processor*), que executam a operação indicada no pedido no contexto do tipo do dispositivo emissor. O processador reencaminha a mensagem de resposta por uma fila de mensagens de saída (i.e. *OUT*), para o adaptador que responde ao dispositivo emissor (100).

O primeiro pedido indica a intenção do dispositivo de se registar e de receber o seu comportamento, podendo este ser representado numa lógica total ou parcial das operações a executar. Contudo é possível que o dispositivo altere a sua dependência face à nuvem computacional durante o seu ciclo de vida, por exemplo, devido a alteração das condições de funcionamento da rede de comunicação.

Sendo assim, um dispositivo pode funcionar de dois modos distintos. Autonomamente, não beneficiando dos recursos computacionais da nuvem para o processamento das suas operações, ou com regras de operação/negócio processadas pela nuvem, residindo opcionalmente no dispositivo ou na nuvem as camadas selecionadas para o efeito. O modo de funcionamento é decidido durante o tempo de vida do dispositivo e pode ser alterado durante o mesmo, de acordo com os parâmetros indicados no capítulo 3.2.

Naturalmente pode existir uma situação excecional, no contexto da qual o dispositivo apenas inclui a lógica para realizar o primeiro pedido à nuvem, e como resposta obtém o código da sua lógica local, ou seja, o dispositivo pode iniciar-se com o código mínimo e obter da nuvem o restante, permitindo a versatilidade do tipo de dispositivo que pode desempenhar. Contudo, a dimensão do código inicial residente no dispositivo fica a cargo do administrador do sistema, que determina qual é a lógica alojada na nuvem e/ou localmente no dispositivo.

De salientar que é totalmente transparente para o dispositivo a existência do adaptador (i.e. *adapter endpoint*). Na perspetiva do dispositivo, este apenas conhece o endereço de um terminal, para o qual envia os pedidos de execução de operações, que responde com o comportamento a ter na próxima operação, consoante o modo de funcionamento do dispositivo.

3.2 Modos *online* e *offline*

Nesta secção apresenta-se a metodologia a adotar no desenvolvimento de dispositivos, de modo a que os dispositivos tirem partido da lógica balanceada, ou seja, transitem entre os modos *online* e *offline*. Essa transição sujeita-se às necessidades do momento do dispositivo, ou seja, pode torná-los dependentes dos recursos da nuvem computacional para a execução de operações que constituem a sua lógica ou autónomos face à nuvem, respetivamente.

No ato de registo, o dispositivo envia para a nuvem um primeiro pedido, com o comando de operação de registo. Este pedido permite registar/autenticar o dispositivo e, simultaneamente obter o código associado a essa operação. Naturalmente, o código retornado é específico de cada tipo de dispositivo.

Caso o comando anterior represente parte do código do tipo do dispositivo (modo *online*) admite-se igualmente um comando *offline*, que retorna a totalidade do código para o dispositivo, tornando-o autónomo face à Internet e à nuvem computacional (modo *offline*).

Nesta proposta admite-se que o dispositivo transita de modo, *online* e *offline*, ao longo do seu tempo de vida, por exemplo, devido a alterações das condições de funcionamento da rede e/ou simplesmente para obter novas versões do código adequado a mudanças da sua lógica de negócio. Esta transição é possível alternando entre os comandos definidos para o efeito, ou seja, um para indicar que pretende apenas obter parte do seu código e continuar a depender da nuvem para a execução das operações (modo *online*), e outro para descarregar a totalidade, garantindo a sua autonomia (modo *offline*).

Naturalmente que a natureza e o valor dos comandos ficam dependentes da implementação realizada.

Para ambos os modos e para o ato de registo do dispositivo, o código retornado deve cumprir o formato, designado por *RegisterResponse*, apresentado na Figura 14.

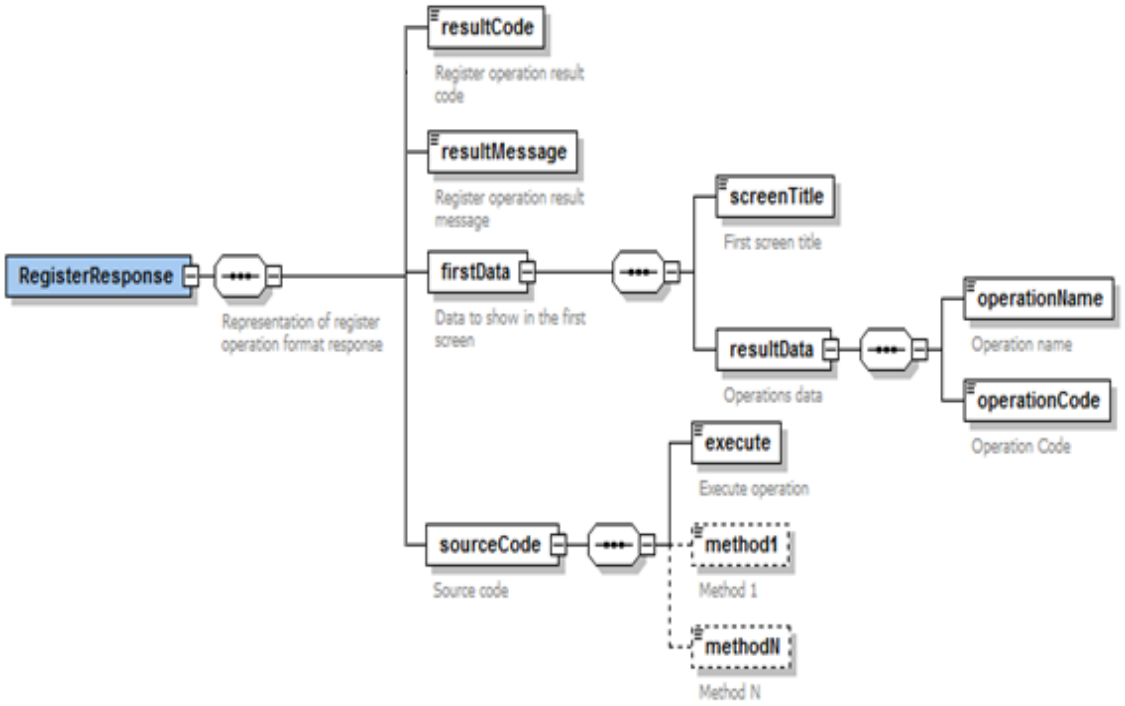

Figura 14 - Formato *RegisterResponse*

Este formato é constituído por um conjunto de membros que permitem enviar a parte ou a totalidade da lógica para o dispositivo respetivo. A Tabela 3 apresenta-os, indicando o significado. A obrigatoriedade de cada um dos membros está indicada na Figura 14, sendo que a linha tracejada indica que se trata de um membro facultativo.

Membro	Significado
resultCode	Código do resultado da execução da operação de registo.
resultMessage	Mensagem do resultado da execução da operação de registo.
firstData	Dados a apresentar no primeiro ecrã.
• *screenTitle*	Título do primeiro ecrã.
• *resultData*	Operações do primeiro ecrã.
○ *operationName*	Nome da operação.
○ *operationCode*	Código que identifica a operação.
sourceCode	Comportamento das operações do primeiro ecrã.
• *execute*	Método obrigatório que inclui o comportamento das operações.
• *method 1…n*	Métodos que complementam o método *execute.*

Tabela 3 - Membros do formato *RegisterResponse*

3.3 Terminais da arquitetura

Nesta secção apresenta-se a informação relativa aos terminais da arquitetura proposta (Figura 13), nomeadamente, o dispositivo, o adaptador, a aplicação para a gestão dos tipos e respetivos dispositivos. Para obter informação complementar ver "Estudo do paradigma computação em nuvem" por Carina Gomes (98).

3.3.1 Dispositivo

A presente secção tem como objetivo caraterizar o tipo de dispositivo suscetível de ser integrado na arquitetura (Figura 13). O dispositivo deve ser autónomo, com capacidade de execução local e de comunicação através da Internet. Adicionalmente, o dispositivo ainda deve dispor de um navegador de Internet (i.e. *browser*).

O dispositivo, materializado sob a forma de uma aplicação móvel, é constituído por um conjunto de camadas, ilustradas na Figura 15, que devem favorecer a coesão, a modularidade e a reutilização das mesmas entre os diversos tipos de dispositivos.

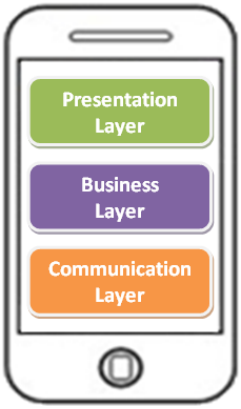

Figura 15 - Camadas do dispositivo

O modelo de camadas, apresentado na Figura 15, é constituído por três camadas, nomeadamente, a de apresentação (i.e. *Presentation Layer*), de lógica/negócio (i.e. *Business Layer*) e de comunicação com a nuvem computacional (i.e. *Communication Layer*).

A camada de apresentação apresenta e gere a informação manipulada na interface do utilizador, sendo esta específica de cada tipo de dispositivo.

A camada intermédia gere a lógica do dispositivo e interpreta os eventos gerados pela interação do utilizador através da camada de apresentação. A camada *Business Layer* é organizada por um conjunto de subcamadas que podem estar alojadas tanto no próprio dispositivo, como na nuvem. Dependendo da implementação das camadas, a dependência/autonomia do dispositivo face à nuvem é variável.

A restante camada é responsável pela comunicação com a nuvem computacional. Esta interação ocorre quando o dispositivo se pretende registar na nuvem, na mudança de modo, *online* ou *offline,* e quando existe a necessidade de executar alguma operação, específica do tipo do dispositivo, no contexto da nuvem.

Os contratos a serem cumpridos pelo dispositivo são ilustrados na Figura 16.

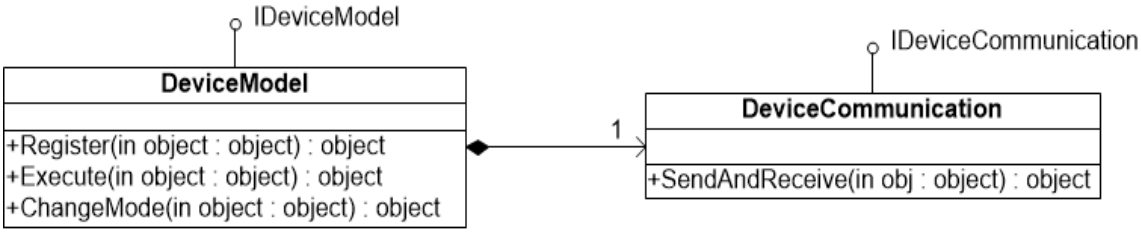

Figura 16 - Diagrama de classes do dispositivo

A camada de lógica inclui um contrato, designado por *IDeviceModel,* que disponibiliza um conjunto de operações indispensáveis para completar a lógica do tipo de dispositivo. A implementação deste contrato é comum a todos os dispositivos, independentemente do seu tipo.

A operação *Register* permite que o dispositivo se registe na nuvem computacional e/ou que receba a informação relativa à próxima operação, ficando portanto dependente da nuvem.

A operação *Execute* é responsável pela execução de operações, que dependendo do tipo do dispositivo, passadas como argumento, a informação relativa à operação a executar (e.g. identificador da operação).

A operação *ChangeMode* envia um pedido para a nuvem com a indicação de que o dispositivo emissor pretende tornar-se autónomo face à nuvem e, portanto, pretende obter a versão completa do *software* com toda a lógica aplicacional, que passa a ser executada localmente.

A camada de comunicação do dispositivo implementa o contrato *IDeviceCommunication.* Este disponibiliza um método, denominado por *SendAndReceive*, cujo comportamento se resume ao envio de pedidos de execução de operações para o adaptador (Figura 13) e consequente espera da correspondente resposta. Esta implementação é independente dos tipos do dispositivo e da nuvem computacional.

3.3.2 Adaptador

O adaptador (i.e. *adapter endpoint*) tem como finalidade receber os pedidos enviados pelos dispositivos e encaminhá-los para a fila de mensagens da arquitetura, num formato de mensagem específico do SDK do fornecedor da nuvem, apresentada na Figura 13.

Neste adaptador está integrada a ferramenta SDK, disponibilizada pelo fornecedor da nuvem computacional. Esta integração deve-se ao facto do adaptador ser o responsável por comunicar com a fila de mensagens de entrada da arquitetura (i.e. '*IN')* e, portanto, fica dependente da tecnologia alvo em que a fila está implementada.

O comportamento do adaptador passa por obter a partir dos pedidos a informação relevante para preencher os atributos que constituem a mensagem. Posteriormente, o adaptador encaminha-a para a fila de entrada e aguarda a resposta emitida pelos processadores (i.e. *processor*), responsáveis pelo processamento das mensagens, através da respetiva fila. Após ter recebido a resposta, o adaptador extrai da mensagem de resposta o resultado da execução da operação e envia-a para o dispositivo emissor do pedido.

A Figura 17 apresenta o diagrama de interação do adaptador com os restantes terminais da arquitetura, o dispositivo e as filas da nuvem computacional, que ilustra o comportamento descrito.

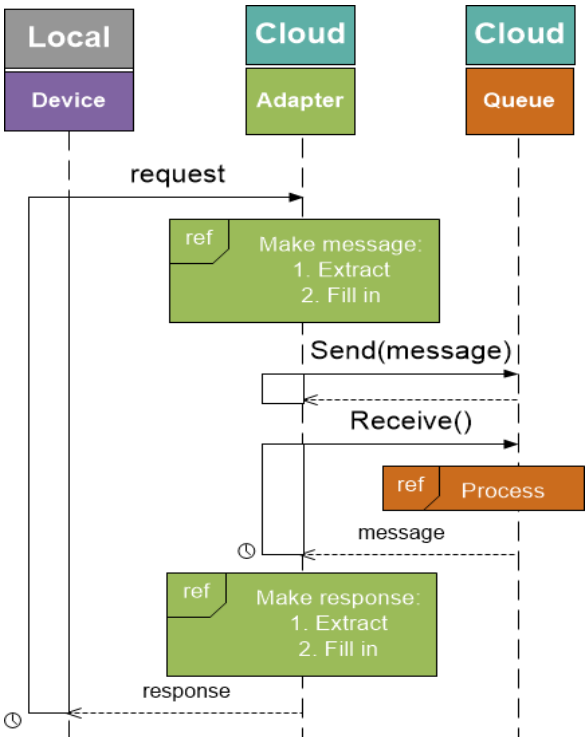

Figura 17 - Diagrama de interação do adaptador

O adaptador implementa o contrato indicado na Figura 18.

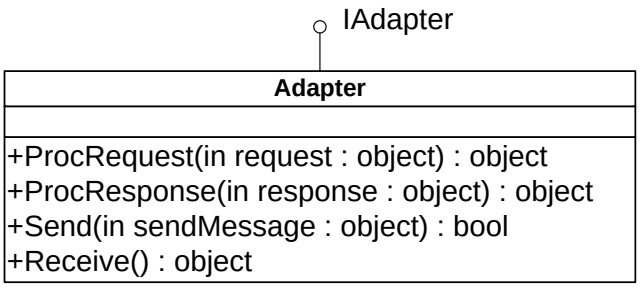

Figura 18 - Interface *IAdapter*

O contrato é constituído por quatro métodos, o *ProcRequest* cuja lógica se resume à instanciação e preenchimento da mensagem, de acordo com os parâmetros recebidos no pedido emitido pelo dispositivo; o *Send* envia a mensagem para a fila de mensagens de entrada; o *Receive* que aguarda pela resposta da nuvem, recebida através da fila de saída e, por fim, o método *ProcResponse* que a partir da mensagem de resposta extrai a informação relevante para construir a resposta para o dispositivo emissor da mensagem de entrada.

A resposta enviada para o dispositivo cumpre o formato *OperationResponse,* apresentado na Figura 19.

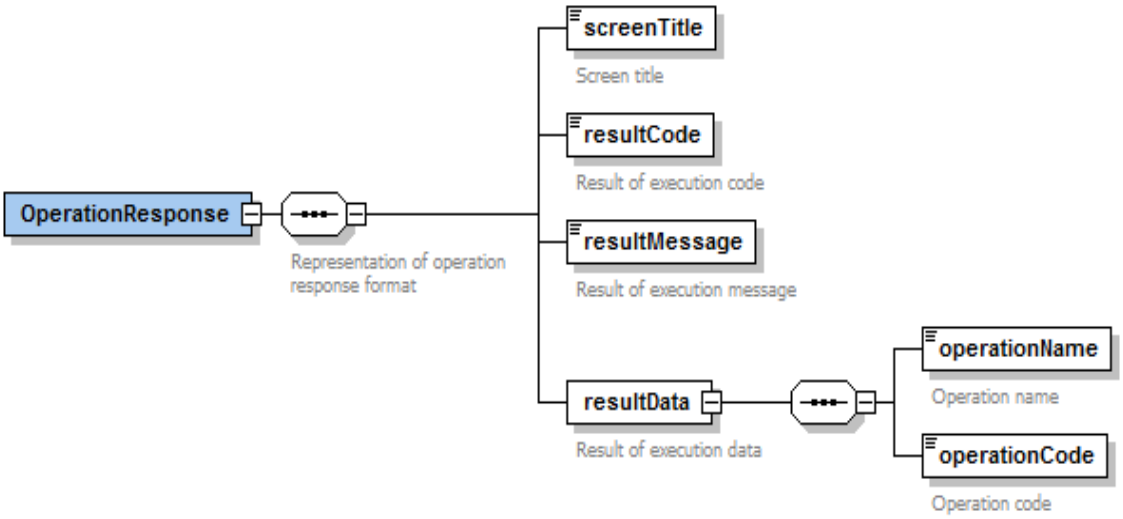

Figura 19 - Formato *OperationResponse*

O formato é organizado pelo título do ecrã (i.e. *screenTitle*), pelo estado da execução (i.e. *resultCode*), a mensagem associada, 'sucesso' ou a mensagem em caso de erro (i.e. *resultMessage*) e o resultado da execução da operação no contexto da nuvem (i.e. *resultData*). Este último é um objeto complexo, formado por pares atributo-valor, que correspondem ao nome da operação (i.e. *operationName*) e o respetivo identificador (i.e. *operationCode*). O seu conjunto resume-se às próximas operações do próximo ecrã a ser apresentado ao utilizador, através da camada de apresentação.

A importância do desenvolvimento deste adaptador passa por permitir aos dispositivos, com reduzidas capacidades de processamento e de armazenamento, comunicarem com as filas de mensagens da nuvem, de um modo transparente. As características físicas do dispositivo (i.e. memória, disco e processador) impossibilitam que seja integrado o SDK necessário à comunicação com as filas referidas.

3.3.3 Aplicação de gestão

A aplicação de gestão dos tipos e respetivos dispositivos possibilita ao administrador do sistema registá-los, para que possa ser validada a autenticação dos dispositivos na emissão dos seus pedidos e, consequentemente, à execução das operações dos tipos de dispositivo no âmbito da nuvem computacional.

Para o efeito, essa aplicação deverá constar num servidor alojado, por exemplo, numa nuvem computacional, disponibilizando uma interface que implemente o contrato *IRegisterDeviceApplication* apresentado na Figura 20.

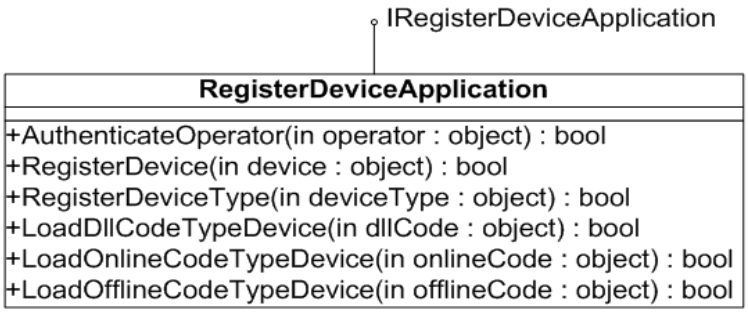

Figura 20 - Interface *IRegisterDeviceApplication*

A interface *IRegisterDeviceApplication* é organizada pelos membros descritos na Tabela 4.

Membros	Descrição	Argumento	Retorno
AuthenticateOperator	Autentica o responsável pelos dispositivos.	Operador responsável pelos dispositivos.	Indica se o resultado da autenticação foi um sucesso.
RegisterDevice	Registo de um dispositivo.	Dispositivo a ser registado.	Indica se o registo do dispositivo aconteceu com sucesso.
RegisterDeviceType	Regista um tipo de dispositivo.	Tipo do dispositivo a registrar	Indica se o registo do tipo do dispositivo ocorreu com sucesso.
LoadDllCodeTypeDevice	Carregamento do código da DLL, referente ao tipo do dispositivo.	Código a ser carregado.	Indica se o carregamento decorreu com sucesso.
LoadOnlineCodeTypeDevice	Carregamento do código *online*, referente ao tipo do dispositivo.	Código a ser carregado.	Indica se o carregamento sucedeu com sucesso.
LoadOfflineCodeTypeDevice	Carregamento do código *offline*, referente ao tipo do dispositivo.	Código a ser carregado.	Indica se o carregamento se realizou com sucesso.

Tabela 4 - Membros da interface *IRegisterDeviceApplication*

3.4 Protocolos

Nesta secção apresentam-se os protocolos de integração, dos dispositivos e dos adaptadores assim como os protocolos de interação entre os terminais da arquitetura (Figura 13).

3.4.1 Protocolo de integração

Para que seja possível a comunicação do dispositivo com a nuvem é necessário respeitar o procedimento seguinte:

a. O administrador de um tipo de dispositivo deve registá-lo na aplicação (secção 3.3.3) disponibilizada para esse efeito e na mesma indicar três componentes de informação, dois para o modo *online* e um terceiro para o *offline,* respetivamente:
    1. **Modo *online*:** Um componente *Dynamic Link Library* (**DLL**) com o código a ser alojado na nuvem e que implemente o contrato denominado *IDeviceType*. Os processadores apresentados na arquitetura limitam-se a invocar dinamicamente o seu único método, *Process*, que consequentemente desencadeia a obtenção do resultado da execução das operações particulares de cada tipo de dispositivo. A assinatura do método referido está apresentada na Figura 21.

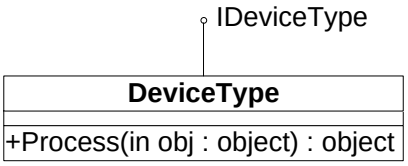

Figura 21 - Contrato *IDeviceType*

    2. **Modo *online*:** um ficheiro com o código a ser retornado para o dispositivo, quando este envia para a nuvem o seu primeiro pedido, ou seja, o de registo. Este deve cumprir o formato apresentado na Figura 14. O código deve complementar o existente na DLL referida no ponto 1.
    3. **Modo *offline*:** Um ficheiro que inclua o código a ser alojado no dispositivo, para o modo *offline*. Este código resume-se à lógica

do tipo de dispositivo, alojada na nuvem, e destina-se aos dispositivos, desse mesmo tipo, que passam a ser autónomos e que, portanto, não dependem da nuvem para a execução das suas operações. O conteúdo desse ficheiro cumpre o formato apresentado na Figura 14 e é enviado para o dispositivo, quando este enviar para a nuvem computacional, o comando correto para esse efeito (secção 3.2).

b. No marcador *AppSettings* do ficheiro de configuração do componente processador, deve ser adicionada uma entrada com os valores nome do tipo do dispositivo e *namespace* da DLL (indicada no ponto anterior a.1), onde estão descritos respetivamente o tipo e os atributos *key* e *value* da entrada.

A integração de um dispositivo pressupõe o registo prévio do tipo do dispositivo respetivo na aplicação disponibilizada para esse efeito (secção 3.3.3).

3.4.2 Protocolo de interação

O protocolo de interação proposto é um melhoramento/refinamento do protocolo apresentado em (98). Esta proposta apresenta a interação, ao nível das camadas, dos quatro terminais da arquitetura (Figura 13), nomeadamente:

a. O dispositivo (i.e. *device*), responsável pela emissão dos pedidos para a execução de operações na nuvem;
b. O adaptador (i.e. *adapter endpoint*) que adequa o formato dos pedidos recebidos para o formato do terminal destinatário;
c. As filas de mensagens (i.e. *queue*) *IN* e *OUT* que são o meio de comunicação entre o adaptador e os processadores (i.e. *processor*);
d. Os processadores que são as instâncias responsáveis pela execução das operações dos dispositivos no contexto da nuvem computacional.

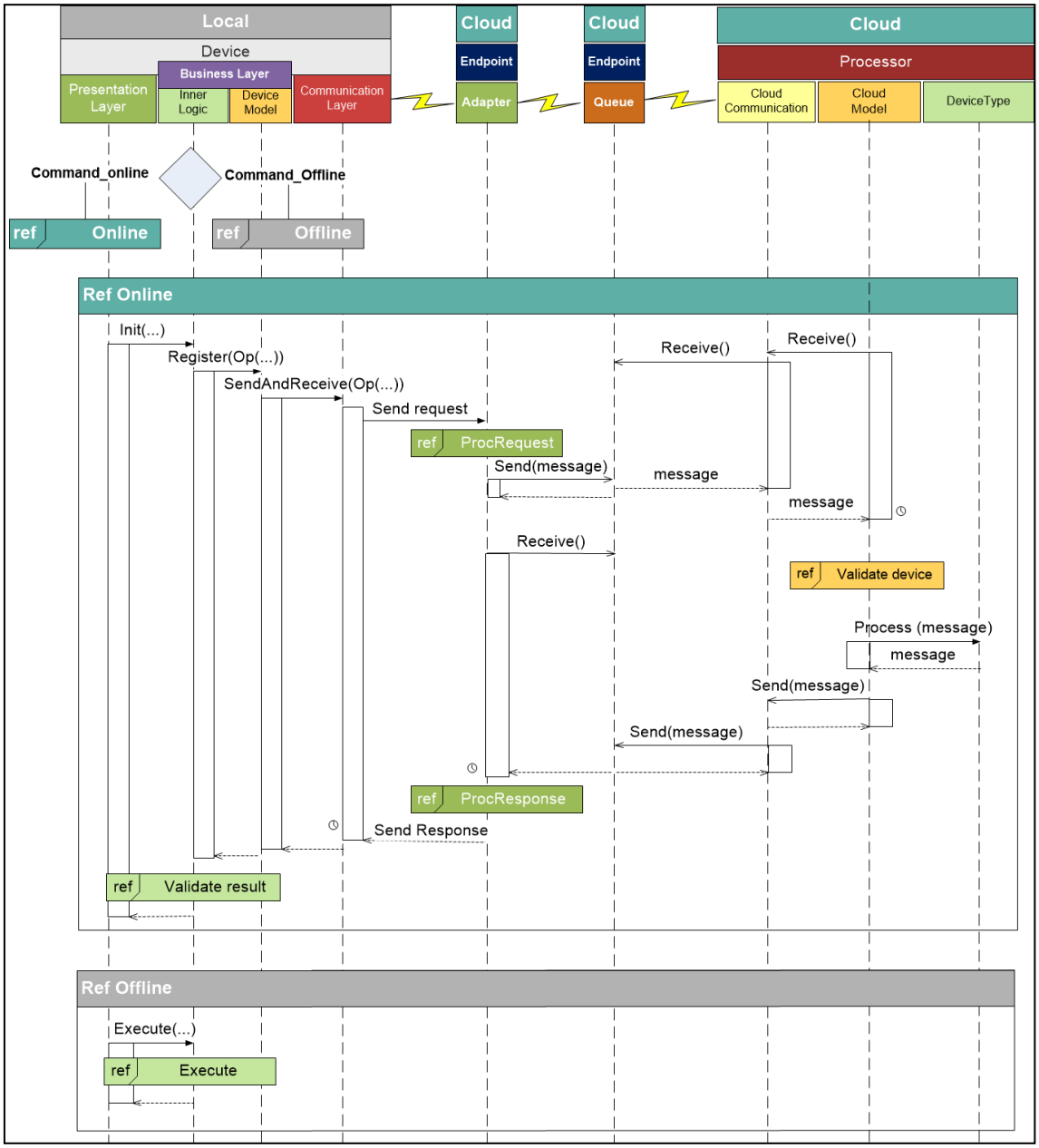

Figura 22 - Protocolo de interação

Após a sua integração, o dispositivo realiza a sua primeira interação com a nuvem computacional. Esta interação inicia-se através do envio de um pedido *Hypertext Transfer Protocol* (**HTTP**) para o adaptador (i.e. *adapter endpoint*), que disponibiliza o seu contrato a partir de uma interface. Este, por sua vez, constrói e armazena o estado do pedido numa mensagem (i.e. *ref ProcRequest*), num formato compatível com a fila de mensagens (i.e. *queue endpoint).*

Posteriormente, os processadores (i.e. *processor*) consomem as mensagens. Por cada mensagem é executada no contexto de uma instância do tipo do dispositivo a operação solicitada.

Na implementação dos tipos dos dispositivos, o comando de registo indica que o dispositivo pretende registar ou apenas ficar em modo *online,* ou seja, dependente da nuvem para a execução das suas operações. O comando *offline* indica que o dispositivo pretende tornar-se autónomo face à nuvem.

Caso a resposta retorne o código a ser alojado no dispositivo, tornando-o independente da nuvem, as próximas operações são executadas no contexto do dispositivo (i.e. *ref Offline*). Caso contrário, mantêm-se as interações entre o dispositivo e a nuvem computacional (i.e. *ref Online*).

Se o identificador da operação for diferente dos anteriores, relativos aos comandos *online* e *offline*, isso implica a execução propriamente dita de uma operação específica do tipo de dispositivo no contexto da nuvem computacional. O retorno para o dispositivo, através da mensagem de resposta, são as próximas operações e/ou o resultado da operação executada.

3.5 Objetos

Nesta secção apresentam-se os objetos que sustentam a arquitetura proposta ilustrada na Figura 13. Os objetos são o formato da mensagem (Figura 23), que permite a troca de informação entre o adaptador (i.e. *adapter endpoint*) e as filas de mensagens (i.e. *queue*), bem como os objetos de dados, que suportam a camada de dados na nuvem. Esta camada é constituída por um objeto que sustenta a informação dos tipos dos dispositivos (Figura 24) e outro para o registo dos mesmos na aplicação desenvolvida para esse efeito (Figura 25).

3.5.1 Mensagem

Na presente secção é apresentado o modelo das mensagens envolvido na interação entre o adaptador e as filas de mensagens da arquitetura apresentada na Figura 13.

As mensagens de ambas as filas da arquitetura, de entrada (i.e. *IN*) e de saída (i.e. *OUT*) de mensagens, respeitam o mesmo modelo. O seu conteúdo informa o destinatário, adaptador (i.e. *adapter endpoint*) e o processador (i.e. *processor*), da informação relevante de cada pedido e do respetivo resultado da operação executada.

A Figura 23 apresenta o modelo das mensagens designado por *Message.*

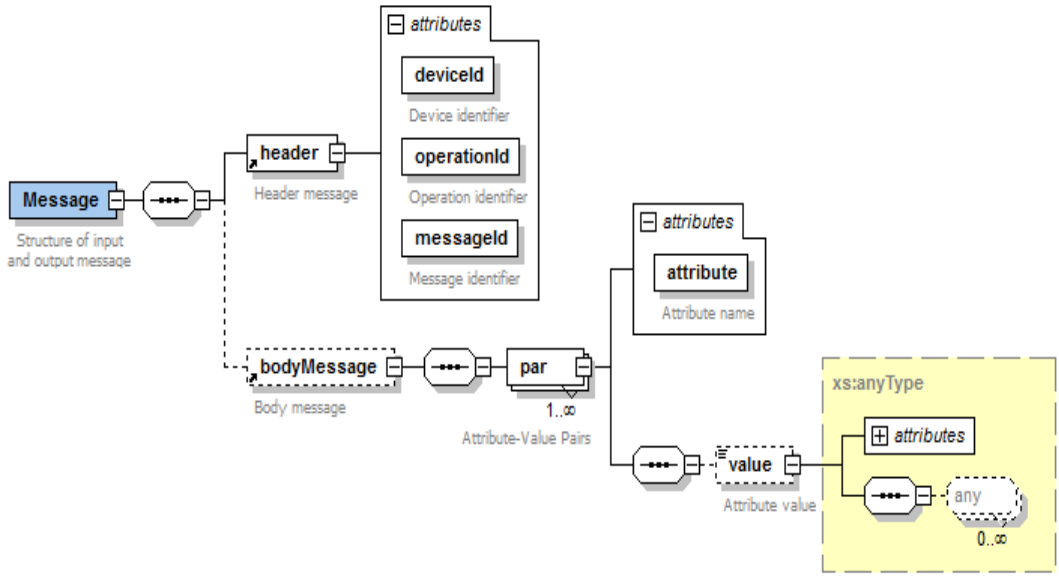

Figura 23 - Objeto *Message*

Um objeto *Message* tem como atributos o identificador do dispositivo emissor (i.e. *deviceId*), o identificador da operação (i.e. *operationId*), o identificador da mensagem (i.e. *messageId*) e o atributo *bodyMessage*. O conteúdo deste último tem caráter opcional. Se for utilizado para enviar um pedido de execução de uma operação para a nuvem, e caso não existam argumentos a serem passados, o atributo referido não tem conteúdo. Por outro, se o formato for utilizado para organizar a informação retornada da nuvem, correspondente ao resultado de uma operação *offline* ou de registo (secção 3.2), este atributo armazena o código a ser alojado no dispositivo, no formato apresentado na Figura 14.

3.5.2 Objetos de dados

Nesta secção são apresentadas as estruturas dos objetos de dados que constituem a camada de dados da arquitetura (Figura 13), no contexto da nuvem computacional.

O objeto *Register* destina-se a armazenar a informação de configuração dos dispositivos registados. A estrutura apresentada na Figura 24 é organizada pelos seguintes atributos: identificador do dispositivo (i.e. *deviceId*), o operador responsável pelo dispositivo (i.e. *operatorId*), o tipo do dispositivo (i.e. *deviceTypeId*), a data do seu registo na aplicação (i.e. *registrationDate*) e a indicação do modo em que o dispositivo se inicia (i.e. *online* ou *offline*), através do atributo *isOnline.*

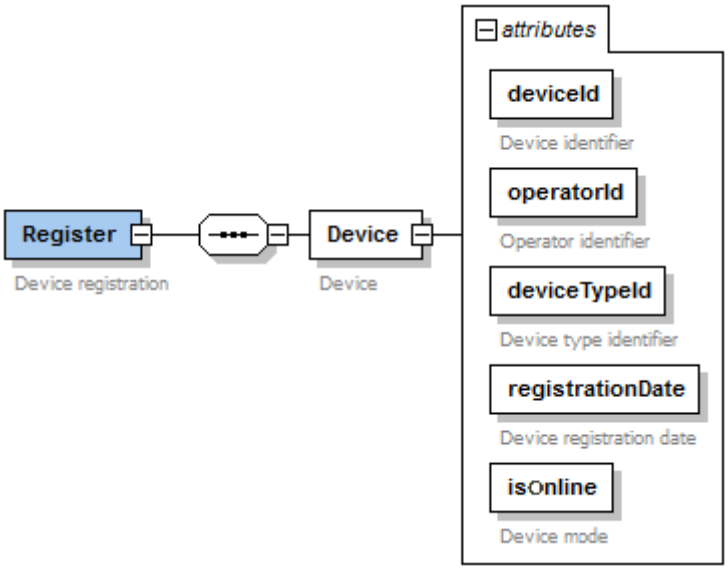

Figura 24 - Objeto de dados *Register*

A informação incluída neste objeto é utilizada pelas instâncias de processadores (i.e. *processor*) que fazem a autenticação dos dispositivos, quando estes enviam pedidos para a nuvem computacional, com o intuito de que sejam executadas as operações indicadas nos referidos pedidos.

Por fim, o objeto de dados, designado por *TypeLogic*, associa aos identificadores do operador (i.e. *operatorId*) e do tipo de dispositivo, um objeto, designado por *sourceCode.* Este último armazena a restante lógica que complementa com a existente localmente no dispositivo. Assim, quando um

dispositivo envia um pedido com o comando *offline* (secção 3.2), este recebe uma resposta que inclui o formato do objecto *sourceCode* (Figura 25).

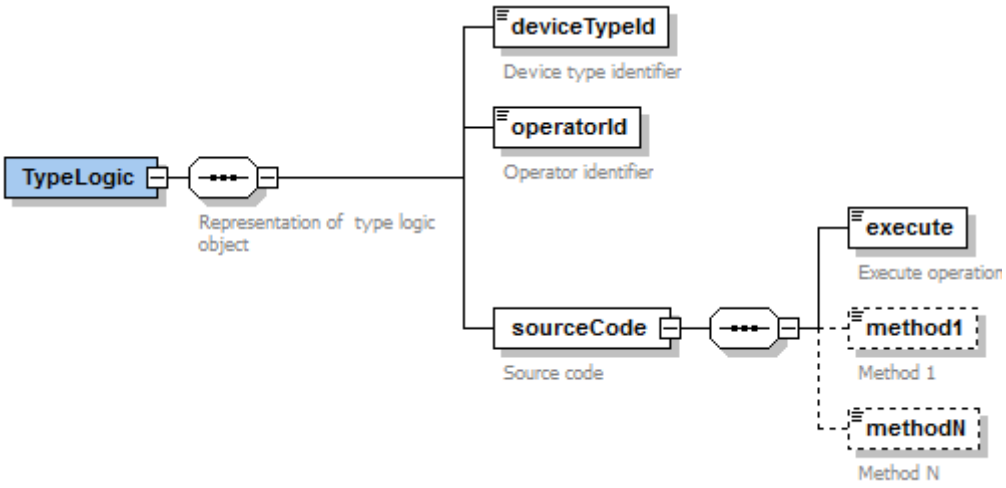

Figura 25 - Objeto de dados *TypeLogic*

No modelo de dados, o objeto *TypeLogic* resume-se aos seguintes membros: o atributo identificador do tipo do dispositivo (i.e. *deviceTypeId*) e o identificador do responsável pelo tipo, associados a um objeto designado por *sourceCode,* que, por sua vez, contém um método obrigatório *execute,* que contém a lógica, e que recorre, opcionalmente, a métodos auxiliares, designados por *method1…n.*

3.6 Considerações finais

Relativamente à proposta, apresentada neste capítulo, destaca-se o seguinte:

a. O desenvolvimento de um tipo de dispositivo é feito recorrendo a tecnologias *web,* mais concretamente, à linguagem *JavaScript* e, portanto, limita o número de plataformas de desenvolvimento móvel que podem ser utilizadas*;*

b. Sendo uma proposta *cross-platform*, ou seja, transversal a várias plataformas móveis (e.g. *Android*) não é adequada para o desenvolvimento de aplicações nativas;

c. A implementação do adaptador (secção 3.3.2) é dependente da tecnologia da nuvem computacional.

A avaliação da solução proposta é efetuada recorrendo à implementação de um projeto *cross-platform,* constituído por um simulador móvel de um dispositivo de bilhética, de um adaptador e de uma aplicação de gestão.

Os requisitos do dispositivo foram indicados pela empresa *Link* (8) e o adaptador foi desenvolvido de modo a se comportar como o intermediário entre o dispositivo e a nuvem computacional WA, sendo esta a responsável pela execução das operações que constituem a lógica dos dispositivos. A implementação da nuvem foi realizada em (100), contudo, esta sofreu alguns melhoramentos, e estes são igualmente apresentados no presente capítulo.

4.1 Instância da arquitetura

Nesta secção descreve-se uma instância da arquitetura de referência (Figura 13), no âmbito da nuvem computacional WA (Figura 26). As camadas de lógica do dispositivo (*Layer 1, Layer 2* e *Layer 3*), que constituem a *Business Layer* (Figura 15), estão alojadas no dispositivo ou na nuvem computacional, dependendo da implementação do tipo realizada.

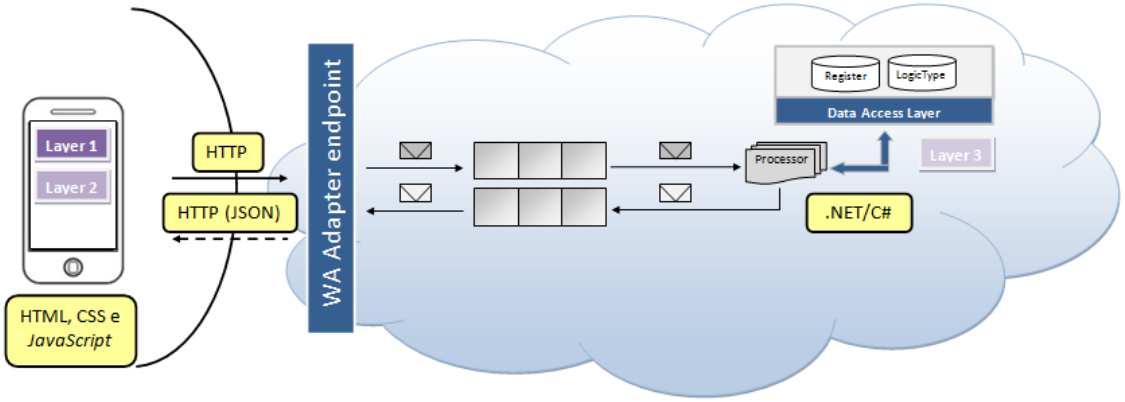

Figura 26 - Instância da arquitetura no âmbito da plataforma WA

A interação entre os vários terminais da arquitetura baseia-se no envio de um pedido HTTP, pelo dispositivo para o adaptador WA (i.e. WA *Adapter endpoint*), enviando na propriedade *queryString* os parâmetros necessários (Figura 23) à execução da operação na nuvem computacional.

Posteriormente, o adaptador, implementado na tecnologia WA, obtém os parâmetros da propriedade *queryString* do pedido recebido e constrói uma mensagem, do tipo específico do WA, *BrokeredMessage* (101), e reencaminha-a para a fila de mensagens de entrada, uma WA *Service Bus Queue* (16). As instâncias de processadores (102) recebem as mensagens, processam-nas e devolvem uma mensagem de resposta, igualmente do tipo *BrokeredMessage,* para a fila de mensagens de saída, uma *Service Bus Topic* (103).

Por fim, o adaptador recebe a mensagem de resposta, desencapsula os parâmetros necessários a construir a resposta para o dispositivo (Figura 23) e envia uma resposta HTTP, com esses parâmetros, no formato apresentado em Figura 19. Salienta-se que caso se trate do primeiro pedido emitido pelo dispositivo, esse pedido e a respetiva resposta cumprem o contrato indicado em Figura 14.

4.1.1 Dispositivo

A comunicação do dispositivo com a nuvem computacional é feita através de pedidos HTTP a um adaptador (i.e. *adapter endpoint*) (Figura 13).

O dispositivo genérico implementado cumpre o contrato anteriormente mencionado (Figura 16). O código do dispositivo é organizado segundo um modelo híbrido (secção 2.2.3), ou seja, combina código nativo e de linguagens de programação da *web* (HTML, CSS e *JavaScript*). A complexidade e tipo de validações a serem realizadas em cada ecrã foram os motivos da referida separação.

Salienta-se que um dispositivo não conhece inicialmente o seu tipo, apenas o fica a conhecer após o envio do primeiro pedido, o de registo. Como resposta obtém o código que lhe permite realizar os pedidos subsequentes, para a realização das operações que serão executadas no âmbito na nuvem. Esta opção de implementação possibilita a diversidade do comportamento que o dispositivo pode ter, de acordo com o código devolvido pela nuvem.

A implementação foi realizada recorrendo a uma plataforma de desenvolvimento (*cross-platform*), designada por IBM *Worklight* (secção 2.3.4). A decisão sobre o uso desta plataforma é justificada pelo facto de já existir experiência profissional anterior, onde se constatou a importância de manipular padrões de aplicações móveis que facilitam e agilizam a implementação da solução proposta.

A aplicação gerada na sequência da fase de desenvolvimento tornou possível a execução do demonstrador em dispositivos móveis com diferentes sistemas operativos (e.g. *Android* (58), *Windows Phone* (104), *Apple iOS* (105), *BlackBerry* (61)).

O tipo do dispositivo implementado cumpre o fluxo de ecrãs apresentado na Figura 27, fornecido pela empresa *Link* (8). Este fluxo resume-se à compra ou recarregamento de títulos de transporte, designadamente por *PreLoadedMobileTicketing*.

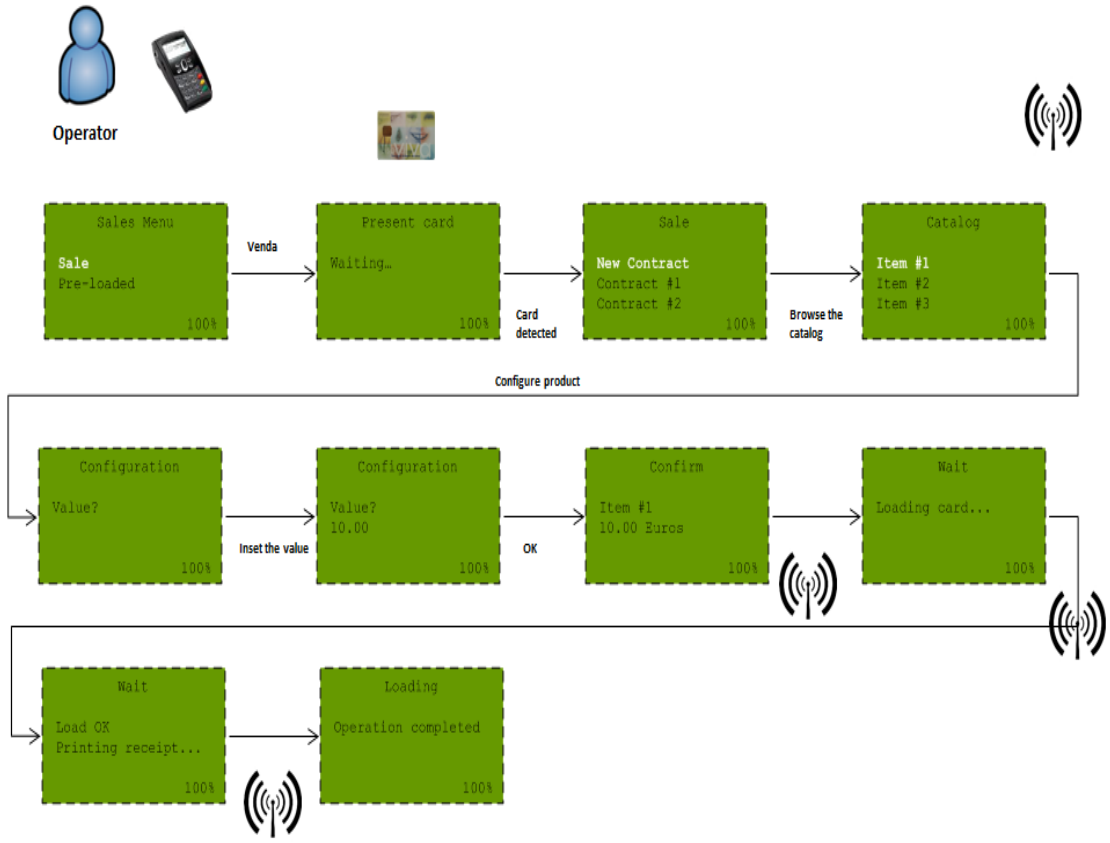

Figura 27 - Fluxo de ecrãs do tipo *PreLoadedMobileTicketing*

A Tabela 5 descreve cada um dos ecrãs bem como o local do seu alojamento, localmente no dispositivo ou no âmbito da nuvem computacional. A sua identificação é feita de acordo com o título de cada ecrã (Figura 27).

Ecrã	Descrição	Local
Sales Menu	Apresenta as primeiras operações.	Nuvem
Present Card	Simula a leitura do cartão.	Dispositivo
Sale	Apresenta as operações após a leitura do cartão.	
Catalog	Após a seleção da opção "*New Contract*" no ecrã anterior, são apresentadas as operações associadas à opção selecionada.	Nuvem
Configuration	Neste ecrã é questionado ao utilizador se aceita o valor associado à realização de um novo contrato.	Dispositivo
Confirm	É pedida a confirmação das opções tomadas nos ecrãs anterior.	Nuvem
Wait	Simulação de espera e carregamento do cartão.	Nuvem
Loading	Notifica o utilizador de que a operação ficou completa. Após este ecrã, o fluxo inicia-se apresentando o primeiro ecrã.	Nuvem

Tabela 5 - Descrição dos ecrãs do fluxo implementado

Tal como apresentado na Figura 15, o dispositivo é organizado em três camadas, a de apresentação (i.e. *Presentation Layer*), a de lógica (i.e. *Business Layer*) e a de comunicação (i.e. *Communication Layer*) com a nuvem computacional.

A camada de apresentação foi implementada recorrendo a tecnologias *web*, nomeadamente, *JavaScript*, CSS3 e HTML5, utilizando a plataforma *Worklight* (capítulo 2.3.4). A aprendizagem e a utilização desta plataforma permitiram a instalação da aplicação desenvolvida em vários dispositivos com diferentes sistemas operativos, o que demonstra a portabilidade da solução móvel. A camada referida está ilustrada na Figura 28 e apresenta o fluxo de ecrãs implementados do tipo *PreLoadedMobileTicketing* (Figura 27).

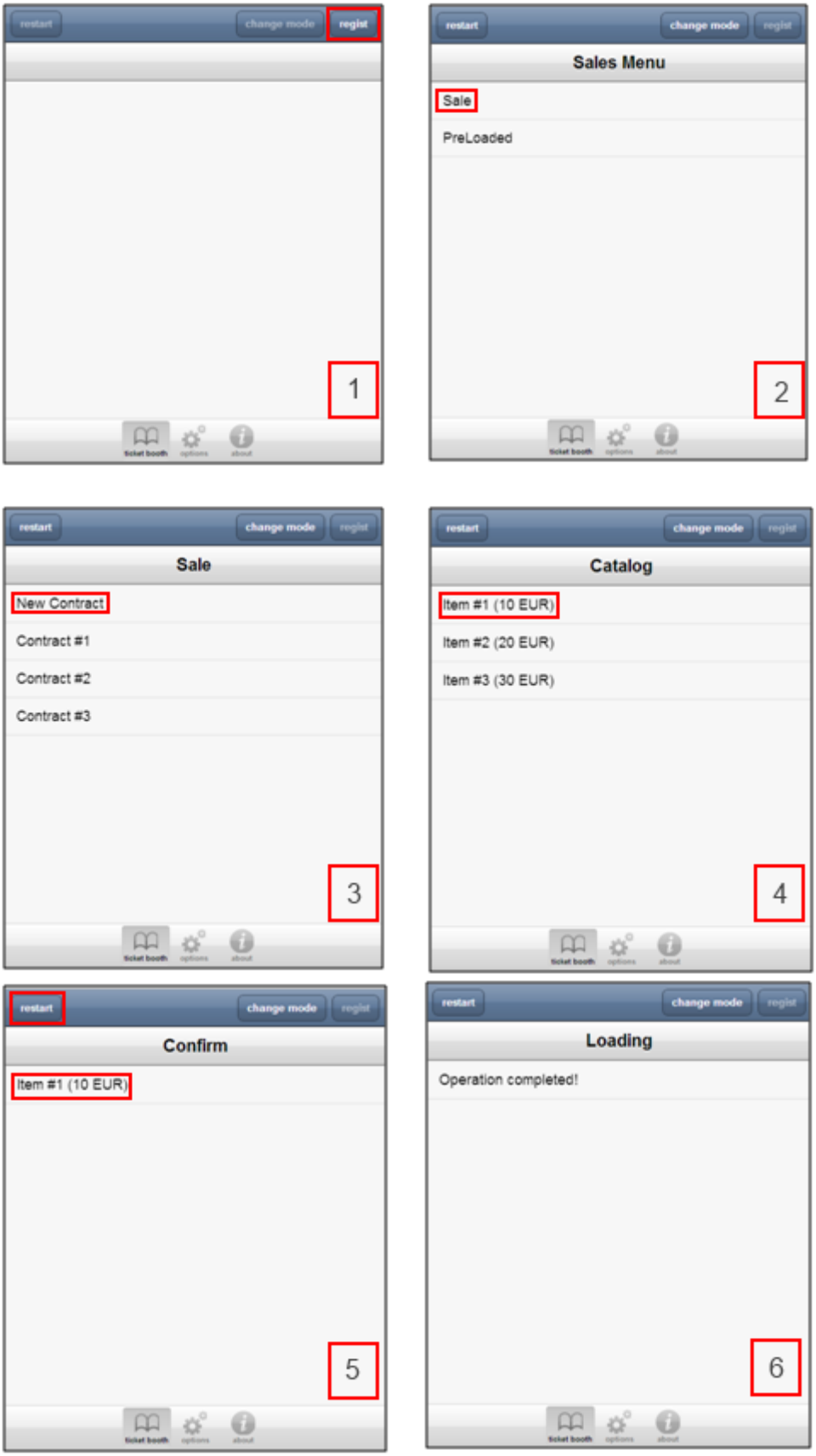

Figura 28 – Estados do dispositivo do tipo *PreLoadedMobileTicketing*

A Figura 28 apresenta seis estados do dispositivo desenvolvido, referentes à venda de um título de transporte, nomeadamente:

1. Apresenta o estado inicial da aplicação no dispositivo. Nesta fase pressiona-se o botão de registo (i.e. *regist*), o qual despoleta um pedido de registo à nuvem, de modo a obter como resposta, a informação relativa ao próximo ecrã e o código com a lógica associada.

2. O ecrã *Sales Menu* apresenta duas operações ao utilizador, nomeadamente, um pré-carregamento e a venda de um título de transporte. Pressionando a opção de venda, despoleta o envio de uma mensagem para a nuvem, a qual retorna a informação do ecrã *Sale,* assinalado com o número três.

3. O ecrã *Sale* apresenta quatro opções de venda de títulos de transporte ao utilizador, nomeadamente, a criação de um novo contrato ou a escolha de um dos três contratos anteriormente comprados. A escolha pelo novo contrato, despoleta o envio de uma mensagem, o que provoca o retorno da informação relevante para a apresentação do ecrã *Catalog,* indicado com o número quatro.

4. O ecrã *Catalog* apresenta as várias alternativas de compra de um novo contrato ao utilizador. Pressionando a opção "*Item #1* (10 EUR)" é enviada uma mensagem, cuja resposta inclui a lógica para a apresentação do ecrã *Confirm* (assinalado com o número cinco).

5. O ecrã *Confirm* dá a possibilidade de confirmar ou cancelar a seleção da opção "*Item #1* (10 EUR)". Caso o utilizador pretenda confirmar, ele deve pressionar a opção "*Item #1* (10 EUR)" apresentada no ecrã. Esta ação despoleta o envio de uma mensagem, cuja resposta apresenta o ecrã *Loading,* assinalado com o número seis. Caso pretenda cancelar, o utilizador deve pressionar o botão *restart*, o qual reinicia o processo de venda de títulos de transporte, ou seja, apresenta o ecrã *Sales Menu,* indicado com o número dois.

6. O ecrã *Loading* informa o utilizador que a compra do novo contrato de um título de transporte foi realizada com sucesso.

Além da sequência de ecrãs apresentada na Figura 28, o utilizador do dispositivo tem acesso às seguintes funcionalidades:

a. O botão *change mode* emite um pedido para a alteração de modo, ou seja, comuta o dispositivo entre os modos *online* e *offline* (secção 3.2).

b. O botão *ticket booth* apresenta o ecrã atual, por exemplo, um dos apresentados na Figura 28.

c. O botão *options* permite gerir configurações do dispositivo (e.g. endereço do adaptador (i.e. *adapter endpoint*)).

d. O botão *about* apresenta informação sobre o autor da aplicação.

A camada de lógica do dispositivo foi desenvolvida recorrendo à linguagem *JavaScript.* A sua constituição engloba a lógica dos ecrãs alojados localmente, que estão identificados na Tabela 5. A lógica local foi obtida na resposta do primeiro pedido emitido pelo dispositivo, o pedido de registo. A restante lógica, referente às operações alojadas na nuvem, é obtida realizando pedidos HTTP à interface do adaptador. Este, por sua vez, após a sua execução, retorna para o dispositivo uma resposta HTTP, com os parâmetros referidos na secção 3.5.1, num formato JSON.

O formato JSON da resposta HTTP, enviada do adaptador para o dispositivo, cumpre o formato apresentado na Figura 14 ou Figura 19, dependendo da operação executada no contexto da nuvem, a de registo/mudança de modo ou uma operação específica do tipo do dispositivo, respetivamente.

Note-se que como a lógica existente no dispositivo e a alojada na nuvem computacional são complementares, a responsabilidade de gerar a resposta em formato JSON é do implementador do tipo de dispositivo. O adaptador apenas encapsula na resposta HTTP, os parâmetros indicados no capítulo 3.5, incluindo o objeto JSON que é enviado da nuvem para o adaptador no atributo *bodyMessage* do objeto *Message* (secção 3.5.1).

Por fim, a camada de comunicação com a nuvem, cumpre o contrato apresentado na Figura 21 designado por *IDeviceCommunication.* O método

(i.e. *SendAndReceive*) envia um pedido HTTP para o adaptador e aguarda a sua resposta, durante um período configurável.

4.1.2 Adaptador

O adaptador apresentado na arquitetura proposta (Figura 13) foi implementado a partir de tecnologias .NET (i.e. resume-se a um *handler* (.*ashx*)), utilizando os recursos disponibilizados pela plataforma WA, implementa a lógica ilustrada no diagrama de interação, apresentado na Figura 17 e cumpre o contrato indicado na Figura 18.

O contrato é disponibilizado através de uma interface *Representational State Transfer* (**REST)** (106) e o seu alojamento foi concretizado num componente do WA designado por *Cloud Service Web Site* (107). Este componente está disponível na ligação: http://meic02wsadapter.azurewebsites.net/adapterendpointconnection.ashx?deviceid=0f8fad5b-d9cb-469f-a165-70867728950e&operationId=-1&messageId=111&bodyMessage=1

De notar que os parâmetros passados no URL do pedido, mais concretamente na propriedade *queryString,* com os parâmetros *deviceId, operationId*, *messageId* e *bodyMessage* com os valores respetivos: '0f8fad5b-d9cb-469f-a165-70867728950e', '-1', '111' e '1'. Estes parâmetros são justamente os atributos que constituem a mensagem (capítulo 3.5.1) a ser enviada do adaptador para a fila de mensagens de entrada.

A função do adaptador consiste na conversão dos pedidos dos dispositivos para um formato de mensagem compatível com as filas de mensagens utilizadas na arquitetura e o retorno da mensagem recebida da nuvem, no formato Figura 23, que inclui o formato apresentado em Figura 14 ou Figura 19, dependendo do tipo de operação executada no âmbito da nuvem, a de registo ou uma específica do tipo do dispositivo.

Salienta-se que a implementação do adaptador foi direcionada para uma plataforma específica, logo este fica dependente não só tipo de mensagens a construir (no caso do WA, trata-se do tipo *BrokeredMessage* (108)), como das

interfaces (i.e. filas de mensagens (109)) que permitem a comunicação com os processadores.

4.2 Aplicação de gestão

A aplicação para a gestão dos dispositivos e respetivos tipos apresenta como interface de utilizador a ilustrada em Figura 29 e cumpre o contrato apresentado na Figura 20.

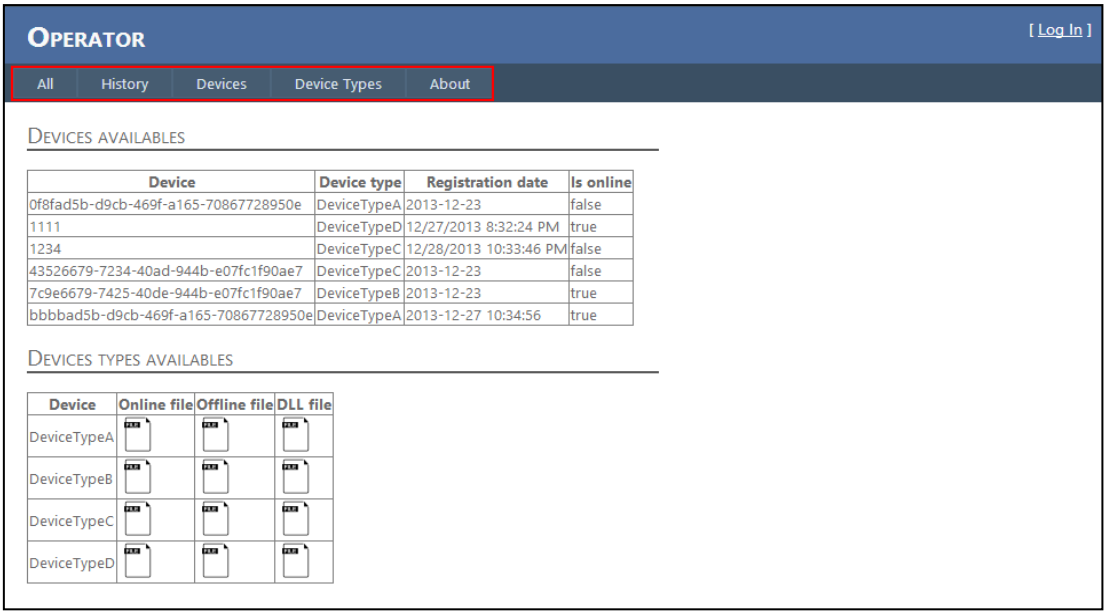

Figura 29 - Interface da aplicação de gestão

Os menus disponíveis e acessíveis aos responsáveis são os indicados na Tabela 6.

Menu	Descrição
Todos (“*All*”)	Apresenta os dispositivos e os seus tipos disponíveis. Para cada um dos tipos é ainda possível realizar o *download* dos ficheiros que os constituem (secção 3.4.1).
Histórico (“*History*”)	Após a seleção do identificador do dispositivo pretendido, é apresentado o histórico das mensagens trocadas por este e pela *cloud.*
Dispositivos (“*Devices*”)	Neste menu, o responsável gere os seus dispositivos.
Tipos de dispositivo (“*Device Types*”)	Neste menu, o responsável gere os tipos dos seus dispositivos.
Sobre (“*About*”)	É apresentada a informação do autor da aplicação.

Tabela 6 - Operações da aplicação do Operador

De notar que a informação apresentada na aplicação é exclusiva do operador autenticado, pelo que apenas o próprio pode gerir os seus dispositivos e respetivos tipos.

A aplicação encontra-se publicada na nuvem computacional WA e está disponível através da ligação: http://meic02wsoperator.azurewebsites.net/.

4.3 Exemplos de cenários de interação

Na presente secção pretende-se apresentar um exemplo de interação entre os terminais: dispositivo, adaptador e nuvem. Os três cenários admitidos cumprem o diagrama *Unified Modeling Language* (**UML**) apresentado na Figura 30.

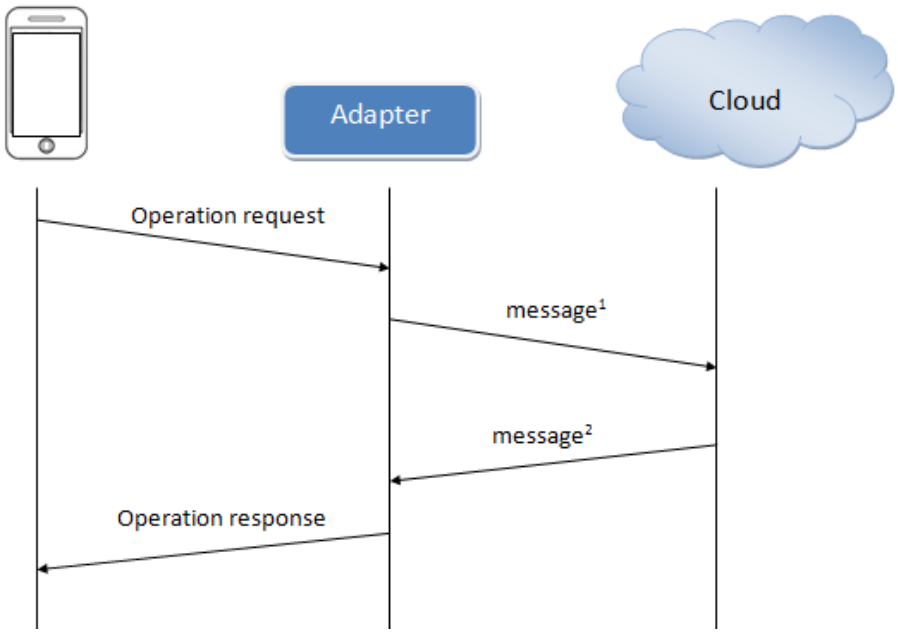

Figura 30 - UML genérico dos cenários de interação entre os terminais

Os cenários são os seguintes:

a. Registo do dispositivo

No primeiro pedido emitido pelo dispositivo para a nuvem, ou seja, o de registo, a informação trocada pelos terminais indicados na Figura 30 são os seguintes:

i. ***Operation request***

Trata-se de um pedido HTTP, com o identificador do dispositivo '0f8fad5b-d9cb-469f-a165-70867728950e', com a operação '-1', com o identificador de mensagem '111' e com o argumento da operação '1':

http://meic02wsadapter.azurewebsites.net/adapterendpointconnection.ashx?deviceid=0f8fad5b-d9cb-469f-a165-70867728950e&operationId=-1&messageId=111&bodyMessage=1

ii. ***Message[1]***

É enviado um objeto *message* (Figura 23), para a fila de mensagens de entrada, com o conteúdo dos parâmetros da *queryString* enviado no *Operation Request.*

iii. ***Message[2]***

É enviado um objeto *message* (Figura 23), para o adaptador com o mesmo identificador do dispositivo ('0f8fad5b-d9cb-469f-a165-70867728950e'), identificador de mensagem ('111') e de operação ('-1'), mas com a propriedade *bodyMessage* com o conteúdo, cumprindo o formato da Figura 14, e preenchido do seguinte modo:

```
{
    resultcode : 0,
    resultMessage : 'Success',
    firstData : {
        screenTitle : 'Sales Menu',
        resultData : [
            {
                operationName : 'PreLoaded',
                operationCode : 1
            },
            {
                operationName : 'Sale',
                operationCode : 2
            }
        ]
    },
    sourceCode : {
        // ... TODO: insert code here
    }
};
```

Figura 31 - Exemplo de resposta de uma operação de Registo

iv. ***Operation response***

O dispositivo recebe a resposta do seu pedido, com o conteúdo referido no *Message[2]*. A propriedade *sourceCode* inclui o código necessário para a próxima operação executar, seja esta, localmente (pois, pode necessitar de pouco processamento) ou na nuvem, devido à sua

complexidade. Como resultado, é apresentado ao utilizador um ecrã, como o apresentado na Figura 28.

b. Execução de uma operação específica do tipo do dispositivo

Na execução de uma operação específica do tipo do dispositivo, de identificador '3', a informação trocada pelos terminais indicados na Figura 30 são os seguintes:

i. ***Operation request***

Trata-se de um pedido HTTP, com o identificador do dispositivo '0f8fad5b-d9cb-469f-a165-70867728950e', da operação '3', da mensagem '112' e sem argumentos: http://meic02wsadapter.azurewebsites.net/adapterendpointconnection.astion.ashx?deviceid=0f8fad5b-d9cb-469f-a165-70867728950e&operationId=-3&messageId=112&bodyMessage=

ii. **Message[1]**

É enviado um objeto *message* (Figura 23), para a fila de mensagens de entrada, com o conteúdo dos parâmetros da *queryString* enviado no *Operation Request.*

iii. ***Message[2]***

É enviado um objeto *message* (Figura 23), para o adaptador com o mesmo identificador do dispositivo ('0f8fad5b-d9cb-469f-a165-70867728950e') e de operação ('3'), mas com a propriedade *bodyMessage* com o conteúdo, cumprindo o formato da Figura 14, e preenchido do seguinte modo:

```
{
    resultcode : 0,
    resultMessage : 'Success',
    screenTitle : 'Catalog',
    resultData : [
        {
            operationName : 'Item #1',
            operationCode : 6
        },
        {
            operationName : 'Item #2',
            operationCode : 7
        },
        {
            operationName : 'Item #3',
            operationCode : 8
        },
        {
            operationName : 'Item #N',
            operationCode : 9
        }
    ]
};
```

Figura 32 - Exemplo de resposta da execução de uma operação específica

iv. ***Operation response***

O dispositivo recebe a resposta do seu pedido, com o conteúdo referido no *Message*[2]. Como resultado, é apresentado ao utilizador um ecrã, como o apresentado na Figura 33.

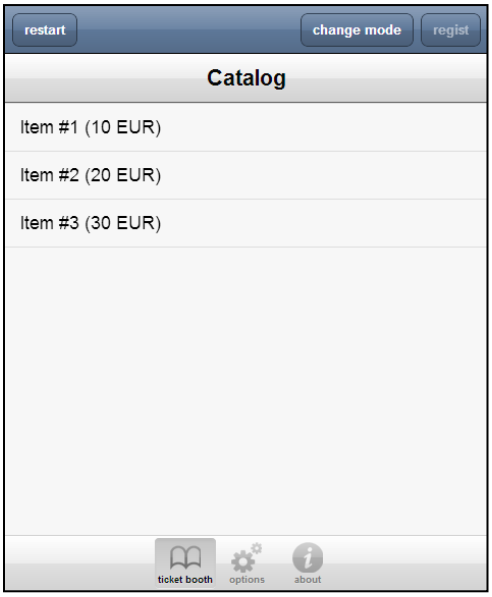

Figura 33 - Dispositivo após a execução de uma operação específica

c. Mudança de configuração, para o modo *offline*

No envio de um pedido para a mudança de configuração, para o modo *offline*, a informação trocada pelos terminais indicados na Figura 30 são os seguintes:

i. ***Operation request***

Trata-se de um pedido HTTP, com o identificador do dispositivo '0f8fad5b-d9cb-469f-a165-70867728950e', da operação '0', do identificador da mensagem '113' e sem argumentos: http://meic02wsadapter.azurewebsites.net/adapterendpointconnection.ashx?deviceid=0f8fad5b-d9cb-469f-a165-70867728950e&operationId=0&messageId=113&bodyMessage=

ii. ***Message[1]***

É enviado um objeto *message* (Figura 23), para a fila de mensagens de entrada, com o conteúdo dos parâmetros da *queryString* enviado no *Operation Request.*

iii. ***Message[2]***

É enviado um objeto *message* (Figura 23), para o adaptador com o mesmo identificador do dispositivo ('0f8fad5b-d9cb-469f-a165-70867728950e''), de operação ('0'), de identificador da mensagem ('113') mas com a propriedade *bodyMessage* com o conteúdo, cumprindo o formato da Figura 14, e preenchido do seguinte modo:

```
{
    resultcode : 0,
    resultMessage : 'Success',
    firstData : {
        screenTitle : 'Sales Menu',
        resultData : [
            {
                operationName : 'PreLoaded',
                operationCode : 1
            },
            {
                operationName : 'Sale',
                operationCode : 2
            }
        ]
    },
    sourceCode : {
        // ... TODO: insert code here
    }
};
```

Figura 34 - Exemplo de resposta de configuração para modo *offline*

iv. ***Operation response***

O dispositivo recebe a resposta do seu pedido, com o conteúdo referido no *Message*[2]. Como resultado, é apresentado ao utilizador um ecrã, como o apresentado na Figura 28. A propriedade *sourceCode* inclui a lógica necessária e complementar, para o dispositivo se tornar autónomo face à nuvem, ou seja, executar as suas operações em modo *offline.*

4.4 Considerações finais

Relativamente às decisões de implementação do projeto, apresentado no presente capítulo, destaca-se:

a. A implementação do adaptador (secção 3.3.2) fica dependente da tecnologia e plataforma escolhidas. Contudo, caso se opte pela tecnologia Java é possível que o adaptador, nestas condições, continue a comunicar com as filas disponibilizadas pela plataforma WA, isto porque esta garante a interoperabilidade entre as tecnologias .NET e Java (110).

b. Caso se verifique que os dispositivos realizam demasiados acessos ao adaptador num curto espaço de tempo, a plataforma WA disponibiliza mecanismos de escalabilidade ajustáveis a métricas indicadas pelo administrador da solução (111).

c. De acordo com as caraterísticas das filas de mensagens disponibilizadas na plataforma WA (112), as operações de envio e de receção de mensagens das filas são assíncronas, segundo uma metodologia de *long-pooling.* Com esta solução, o adaptador fica bloqueado durante um período de tempo o que, consequentemente reduz o número de transações de interação e os custos associados. Durante o período de espera, o dispositivo simula trabalho, através de um componente próprio, apresentado na camada de apresentação.

Atualmente, os dispositivos móveis apresentam desafios em diferentes contextos no mundo das TI. As empresas procuram abordagens de desenvolvimento adequadas para alcançarem os seus objetivos, de modo a resolverem as necessidades de crescimento e de complexidade do mundo móvel atual. Mais concretamente, no contexto dos transportes públicos, os operadores pretendem ter um modo rápido, simples, confiável e de baixo custo, para disponibilizar aplicações aos seus clientes (35).

A solução proposta utiliza uma abordagem baseada em PaaS e foi desenvolvida a plataforma móvel, *Worklight* (secção 2.3.4), tendo sido demonstrada no contexto de serviços de bilheteira alojados parcialmente numa nuvem computacional. Neste contexto ficou também demonstrada a portabilidade da aplicação móvel que simula os dispositivos de bilhética para outras plataformas.

A arquitetura proposta, ilustrada na Figura 13, estabelece a forma como os dispositivos móveis podem comunicar com os serviços (i.e. bilhética), de modo a suportarem os processos de negócio e o desenvolvimento do sistema de bilhética numa abordagem baseada em serviços alojadas numa nuvem computacional.

A arquitetura proposta facilita igualmente o balanceamento da lógica dos dispositivos (aplicação móvel) com a nuvem computacional. Ou seja, o dispositivo móvel pode descarregar o código, suportando-o localmente (modo *offline*), ou interagir com a nuvem computacional, para a execução de operações (modo *online*).

5.1 Conclusão

O objetivo do presente trabalho consistiu no desenvolvimento de uma arquitetura direcionada para a integração de dispositivos móveis, no âmbito da bilhética de transportes, numa nuvem computacional.

Este projeto está associado ao projeto *SmartCITIES Cloud Ticketing* desenvolvido numa parceria entre o ISEL e a empresa Link (8), que tem como principal objetivo avaliar a possibilidade de tornar os dispositivos atualmente existentes no mercado segundo o modelo *fat-client*, em *thin-client,* transportando parte da lógica destes dispositivos de bilhética, para uma nuvem computacional. Esta possibilidade permite uma maior flexibilidade de gestão dos dispositivos, inclusive, na sua implementação, pela reutilização de código entre vários tipos de dispositivos.

Tendo em conta a diversidade de operações no contexto da bilhética de transportes foi desenvolvido um protótipo que cumpre o fluxo de ecrãs que esboça a interação com um dispositivo real (Figura 27).

O objetivo da integração da lógica dos dispositivos de bilhética numa nuvem computacional consiste em alcançar a centralização da lógica de negócios e movimentar a lógica terminal específica dos dispositivos móveis para a nuvem, portanto, reduzindo a complexidade e os custos dos dispositivos terminais. Adicionalmente promove a colaboração e partilha de infraestruturas computacionais entre operadores de transportes em particular numa determinada área geográfica.

5.2 Perspetivas futuras

Finalmente destacam-se como linhas de trabalho futuro as seguintes:

a. Instanciar a arquitetura proposta noutras plataformas de desenvolvimento móvel, como por exemplo, as indicadas no capítulo 2.3, de modo a avaliar a que oferece os componentes que mais se adequam às necessidades de desenvolvimento. Essa escolha deve-se basear no tipo de componentes que são disponibilizados, face aos que são necessários para a sua instanciação, por exemplo, ao custo que está inerente ao uso e à dependência das aplicações face à sua utilização.

b. Recorrer a outras nuvens computacionais para a execução e alojamento da lógica dos tipos de dispositivos, avaliando a que melhor corresponde às necessidades de resposta em tempo real dos dispositivos físicos, nomeadamente, ao nível da qualidade da comunicação e da dependência face ao fornecedor da nuvem computacional que condicionam aa eficácia e eficiência da solução proposta.

c. Apesar do foco do presente trabalho consistir em provar que é possível integrar dispositivos de bilhética numa nuvem computacional, seria interessante o desenvolvimento de outros tipos de dispositivos, com operações mais complexas, de modo a compreender as limitações e melhorar a arquitetura proposta.

1. The Mobile World is. [Online] 2012. [Cited: 02 04, 2013.] http://www.ibm.com/mobile-enterprise/us/en/bin/pdf/IBMmobileopad.pdf.

2. Android. Discover a new flavor of Jelly Bean. [Online] 2013. [Cited: 02 17, 2013.] http://www.android.com/.

3. Windows Phone. Conheça o novo Windows Phone. [Online] 2013. [Cited: 02 17, 2013.] http://www.windowsphone.com/pt-br.

4. iOS. iOS 6. [Online] 2013. [Cited: 02 17, 2013.] http://www.apple.com/br/ios/.

5. Windows Azure. A rock-solid cloud platform for blue-sky thinking. [Online] 2013. [Cited: 02 17, 2013.] http://www.windowsazure.com/en-us/.

6. Google App Engine. What Is Google App Engine? [Online] 2013. [Cited: 02 17, 2013.] https://developers.google.com/appengine/docs/whatisgoogleappengine.

7. Amazon Web Services. Inovação empresarial. [Online] 2013. [Cited: 02 17, 2013.] http://aws.amazon.com/pt/.

8. Link - Gerimos conhecimento, consigo. Link - Gerimos conhecimento, consigo. [Online] 2013. [Cited: 09 07, 2013.] http://www.link.pt/.

9. LEE, V. and SCHNEIDER, H. SCHELL, R. Aplicações móveis: arquitetura, projeto e desenvolvimento. São Paulo : s.n., 2005.

10. Pereira, Pedro. Centro de Cálculo. Programação em Dispositivos Móveis (PDM). [Online] 2012. [Cited: 09 14, 2013.] http://thoth.cc.e.ipl.pt/classes/PDM/1112v/LI61N-MI2N/resources/42.

11. Peter Mell, Timothy Grance. National Institute of Standards and Technology. The NIST Definition of Cloud Computing . [Online] 01 2001. [Cited: 09 15, 2013.] http://pre-developer.att.com/home/learn/enablingtechnologies/The_NIST_Definition_of_Cloud_Computing.pdf.

12. ACM Digital Library. A view of cloud computing. [Online] 2010. [Cited: 09 15, 2013.] http://delivery.acm.org/10.1145/1730000/1721672/p50-armbrust.pdf?ip=85.247.18.119&id=1721672&acc=OPEN&key=BF13D071DEA4D3F3B0AA4BA89B4BCA5B&CFID=361683272&CFTOKEN=85361799&__acm__=1379240944_05e2b155a1f8f4d518e02bd11d733596.

13. Grace Walker. developerWorks. Aspectos fundamentais da computação em nuvem. [Online] 01 13, 2011. [Cited: 09 15, 2013.] http://www.ibm.com/developerworks/br/cloud/library/cl-cloudintro/ .

14. Peter Mell, Tim Grance. Effectively and Securely Using the . [Online] 07 10, 2009. [Cited: 09 15, 2013.] http://gat1.isoc.org.il/conf2010/handouts/Yesha_Sivan.pdf.

15. Working with Worklight, Part 1: Getting started with your first Worklight application. [Online] 2012. [Cited: 02 04, 2013.] http://www.ibm.com/developerworks/mobile/library/mo-aim1206-working-with-worklight-1/index.html.

16. Windows Azure. Service Bus Queues, Topics, and Subscriptions. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://msdn.microsoft.com/en-us/library/windowsazure/hh367516.aspx .

17. Augusto Nunes. Projeto - JEDI - Desenvolvimento - de - Aplicações - Moveis. [Online] 2012. [Cited: 02 04, 2013.] http://www.ebah.com.br/content/ABAAAfLJYAC/projeto-jedi-desenvolvimento-aplicacoes-moveis-java-164-paginas.

18. Vanilson André de Arruda Burégio. UNIVERSIDADE FEDERAL DE PERNAMBUCO GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO CENTRO DE INFORMÁTICA. DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVO MÓVEIS COM .NET. [Online] [Cited: 02 04, 2013.] http://www.cin.ufpe.br/~tg/2003-1/vaab.PDF.

19. oldcomputers. Osborne 1. [Online] 2012. [Cited: 02 04, 2013.] http://oldcomputers.net/osborne.html.

20. Compaq Portable. [Online] 2012. [Cited: 02 04, 2013.] http://oldcomputers.net/compaqi.html.

21. RADIO SHACK TRS-80 and TANDY COMPUTER CATALOGS. [Online] 2012. [Cited: 02 04, 2013.] http://www.radioshackcatalogs.com/computer.

22. Psion Organizer I. The Psion Organiser 1. [Online] 2005. [Cited: 02 04, 2013.] http://archive.psion2.org/org2/psion1/.

23. A Brief History Of Psion's Machines. [Online] 09 14, 2006. [Cited: 02 04, 2013.] http://www.bioeddie.co.uk/Psion/main/psion-machines.htm.

24. Allan, Roy. A History of the Personal Computer: The People and the Technology. 2001. ISBN 978-0-9689108-0-1.

25. Newton MessagePad 100. [Online] 1993. [Cited: 02 04, 2013.] http://applemuseum.bott.org/sections/computers/omp.html.

26. Palm Treo Smartphone & PDA Reviews. [Online] 2002. [Cited: 02 04, 2013.] http://www.palminfocenter.com/news/3628/all-reviews-of-pdas-handhelds-and-smartphones/.

27. Chris Tilley. The History Of Windows CE. [Online] 07 19, 2012. [Cited: 02 04, 2013.] http://www.hpcfactor.com/support/windowsce/.

28. The IBM Rational solution for systems and software engineering. [Online] 2012. [Cited: 02 04, 2013.] http://www-01.ibm.com/software/rational/announce/embedded/.

29. Best Practices Hybrid Mobile Apps. [Online] 2012. [Cited: 02 04, 2013.] http://www.slideshare.net/charlesying/best-practices-hybrid-mobile-native-web-apps.

30. Collaborative Lifecycle Management. [Online] 2012. [Cited: 02 04, 2013.] http://www-01.ibm.com/software/rational/alm/collaborate/.

31. Smythe, Albert Ho and Will. Accelerate Mobile Application Delivery. IBM. [Online] 2012. [Cited: 02 2013, 04.] https://www.ibm.com/developerworks/rational/library/accelerate-software-systems-delivery-spring-launch/accelerate-software-systems-delivery-spring-launch-pdf.pdf.

32. Mike Perrow. Accelerate software and systems deliveryacross the application and product lifecycle. [Online] 06 04, 2012. [Cited: 02 04, 2013.] http://ebookbrowse.com/accelerate-software-systems-delivery-spring-launch-pdf-pdf-d424789797.

33. Monica Basso, Phillip Redman . Critical Capabilities for Mobile Device . [Online] 08 02, 2012. [Cited: 02 04, 2013.] Critical-Capabilities-for-Mobile-Device-Management.pdf].

34. Anyone, Anywhere, Anytime – empowering teams everywhere! [Online] 2012. [Cited: 02 04, 2013.] http://www-01.ibm.com/software/rational/announce/software/lifecycle/.

35. Native, web or hybrid . [Online] 2012. [Cited: 02 04, 2013.] ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf.

36. Getting a better grip on mobile . [Online] 2012. [Cited: 02 04, 2013.] http://public.dhe.ibm.com/common/ssi/ecm/en/tiw14127usen/TIW14127USEN.PDF.

37. Suporte da Mac App Store. Apple. [Online] Apple, 2013. [Cited: 09 14, 2013.] http://www.apple.com/pt/support/mac/app-store/.

38. Google Play Store. Google. [Online] Google, 2013. [Cited: 09 14, 2013.] https://play.google.com/store?hl=pt_PT.

39. BlackBerry World. BlackBerry. [Online] BlackBerry, 2013. [Cited: 09 14, 2013.] http://appworld.blackberry.com/webstore/?lang=pt_br.

40. Eight steps to IBM Worklight mobile application development. [Online] 10 2012. [Cited: 02 04, 2013.] http://www.ibm.com/developerworks/websphere/techjournal/1210_chen/1210_chen.html.

41. Robin Berjon, Steve Faulkner, Travis Leithead. W3C Candidate Recomendation. HTML5. [Online] W3C, 08 06, 2013. [Cited: 09 14, 2013.] http://www.w3.org/TR/html5/.

42. Cascading Style Sheets. Cascading Style Sheets. [Online] W3C, 09 13, 2013. [Cited: 09 14, 2013.] http://www.w3.org/Style/CSS/ .

43. JavaScript Reference. MSDN. [Online] Microsoft, 2013. [Cited: 09 14, 2013.] http://msdn.microsoft.com/en-us/library/ie/yek4tbz0(v=vs.94).aspx.

44. Dojo. Dojo Mobile. [Online] Dojo Foundation, 2012. [Cited: 09 14, 2013.] http://dojotoolkit.org/reference-guide/1.9/dojox/mobile.html.

45. Sencha. Sencha Touch Build Mobile Web Apps with HTML5. [Online] Sencha, 2013. [Cited: 09 14, 2013.] http://www.sencha.com/products/touch.

46. jQuery Mobile. jQuery Mobile: Touch-Optimized Web Framework for Smartphones & Tablets. [Online] jQuery Foundation, 2013. [Cited: 09 14, 2013.] http://jquerymobile.com/.

47. Enterprise Caching in a Mobile Environment. [Online] 12 27, 2012. [Cited: 02 04, 2013.] http://ebookbrowse.com/ib/ibm-redbooks?page=8.

48. PhoneGap. Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript. [Online] PhoneGap, 2013. [Cited: 09 14, 2013.] http://phonegap.com/ .

49. Appcelerator . Mobile innovation is a journey, not a result. [Online] Appcelerator Inc., 2013. [Cited: 09 14, 2013.] http://www.appcelerator.com/.

50. Xamarin. Xamarin - Developer Center. [Online] Xamarin, 2013. [Cited: 09 14, 2013.] http://docs.xamarin.com/.

51. IBM Worklight. IBM Worklight. [Online] IBM, 2013. [Cited: 09 14, 2013.] http://www-03.ibm.com/software/products/us/en/worklight/.

52. Intel Software. Desenvolvendo seu primeiro aplicativo móvel multiplataforma com HTML5. [Online] 08 13, 2012. [Cited: 02 13, 2013.] http://software.intel.com/pt-br/blogs/2012/08/13/desenvolvendo-seu-primeiro-aplicativo-mvel-multiplataforma-com-html5.

53. Laboratório IMobilis Computação Móvel . Avaliação Da Ferramenta PhoneGap – Parte 1. [Online] 11 19, 2012. [Cited: 02 13, 2013.] http://www.decom.ufop.br/imobilis/?p=1459.

54. R Ghatol, Y Patel. Beginning PhoneGap - Chapter 9. [Online] 2012. [Cited: 09 15, 2013.] http://link.springer.com/content/pdf/10.1007/978-1-4302-3904-8_9.pdf#page-2.

55. William Enck,Damien Octeau, Patrick McDaniel. USENIX Security Symposium. A Study of Android Application Security. [Online] 08 2011. [Cited: 09 15, 2013.] https://www.usenix.org/legacy/event/sec11/tech/slides/enck.pdf.

56. Jonathan Stark. Building iPhone Apps with HTML, CSS, and JavaScript: Making App Store Apps. [Online] 01 2010. [Cited: 09 15, 2013.] http://www.google.pt/books?hl=pt-PT&lr=&id=3NQhBTNKJB8C&oi=fnd&pg=PR7&dq=Objective-C+para+IPhone.&ots=j90nv_qp6r&sig=0Z8ZBwpgoFncQIUHvuYK7yzqr6o&redir_esc=y#v=onepage&q=Objective-C%20para%20IPhone.&f=false.

57. Developer Apps for iOS. The World's most advanced mobile operation system. [Online] Apple, 2013. [Cited: 09 14, 2013.] https://developer.apple.com/technologies/ios/.

58. Android. Developer. [Online] Google, 2013. [Cited: 09 14, 2013.] http://developer.android.com/index.html.

59. webOs Deveoper Center. Open webOS. [Online] Hewlett-Packard Development Company, 2013. [Cited: 09 14, 2013.] https://developer.palm.com/.

60. symbian developers. symbian developers. [Online] symbian , 2013. [Cited: 09 14, 2013.] http://symbian-developers.net/.

61. BlackBerry Developer. BlackBerry Developer. [Online] BlackBerry , 2013. [Cited: 09 14, 2013.] http://developer.blackberry.com/.

62. PhoneGap. Supported Features. [Online] 2013. [Cited: 02 13, 2013.] http://phonegap.com/about/feature/.

63. Compromisso Digital. Conhecendo a framework Phonegap. [Online] [Cited: 02 13, 2013.] http://www.compromissodigital.com/2012/09/conhecendo-framework-phonegap.html.

64. jQuery user interface. Development Center. [Online] jQuery Foundation, 2013. [Cited: 09 14, 2013.] http://jqueryui.com/development/.

65. mootools. mootools a compact javascript framework. [Online] mootools, 2012. [Cited: 09 14, 2013.] http://mootools.net/forge/p/mobile.

66. xui. xui. [Online] xui, 2008. [Cited: 09 14, 2013.] http://xuijs.com/.

67. PhoneGap. Image Caching with the HTML5 Canvas. [Online] PhoneGap, 2013. [Cited: 09 14, 2013.] http://phonegap.com/2009/09/24/image-caching-with-the-html5-canvas/.

68. PhoneGap. iOS 5.1 and the embedded UIWebView with Cordova. [Online] PhoneGap, 2013. [Cited: 09 14, 2013.] http://phonegap.com/2012/04/18/ios-5-1-and-the-embedded-uiwebview-with-cordova/.

69. Mobiltec - Mobilidade em negócios - Blog. Desenvolvimento Mobile Multiplataforma com Phonegap. [Online] 10 17, 2011. [Cited: 02 13, 2013.] http://www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-multiplataforma-com-phonegap/.

70. Create & Manage Native Mobile Apps Across Devices. [Online] 2013. [Cited: 02 10, 2012.] http://www.appcelerator.com/platform/titanium-platform/.

71. Appcelerator. Adopting a Mobile Center of Excellence (MCoE) to Become a Mobile-First Enterprise. [Online] 2012. [Cited: 02 10, 2013.] https://pages.appcelerator.com/1.8.12Whitepaper-MobileCenterofExcellence.html#.

72. Benjamin Bahrenberg. Appcelerator Titanium Business Application Development Cookbook. [Online] 06 2013. [Cited: 09 15, 2013.] http://www.google.pt/books?hl=pt-PT&lr=&id=igix9GkI8VIC&oi=fnd&pg=PT7&dq=Arquitecture+Appcelerator+Titanium&ots=nv5FnS-rO9&sig=gl7-mkJh7KlEDQoQHcisOAc0NyQ&redir_esc=y#v=onepage&q=architecture&f=false.

73. Hyeong-Seok Oh. Evaluation of Android Dalvik virtual machine. [Online] 2012. [Cited: 09 15, 2013.] http://dl.acm.org/citation.cfm?id=2388956.

74. C Kerschbaumer, E Hennigan, P Larsen. Towards Precise and Efficient Information Flow Control in Web Browsers. [Online] 2013. [Cited: 09 15, 2013.] http://link.springer.com/chapter/10.1007/978-3-642-38908-5_14#page-2.

75. Appcelerator Titanium 3.0. Transitioning to the New UI Layout System. [Online] 2013. [Cited: 02 10, 2013.] http://docs.appcelerator.com/titanium/3.0/#!/guide/Transitioning_to_the_New_UI_Layout_System-section-30088148_TransitioningtotheNewUILayoutSystem-Overview.

76. appcelerator.com. 4-steps-create-mobile-strategy. [Online] 2011. [Cited: 02 10, 2013.] http://www.appcelerator.com.s3.amazonaws.com/blog/www/files/4-steps-create-mobile-strategy.pdf.

77. Cloud Services. The Cloud Starts Here™: Titanium Cloud Services. [Online] Appcelerator, 2013. [Cited: 09 14, 2013.] http://www.appcelerator.com/cloud/.

78. Simon Berman. Appcelerator. Enterprise Mobility: Cloud Services and its Impact on Mobile Server-Side Development. [Online] 09 10, 2012. [Cited: 02 10, 2013.] http://thinkmobile.appcelerator.com/blog/bid/213930/Enterprise-Mobility-Cloud-Services-and-its-Impact-on-Mobile-Server-Side-Development.

79. Appcelerator . The Cloud Starts Here: Appcelerator Cloud Services (ACS). [Online] 2013. [Cited: 02 10, 2013.] http://www.appcelerator.com/cloud/.

80. Using Cloud Services for Building. [Online] 2012. [Cited: 02 10, 2013.] http://docs.appcelerator.com/titanium/latest/#!/guide/Quick_Start.

81. Xamarin. Xamarin - Create iOS, Android, Mac and Windows in C#. [Online] 2013. [Cited: 04 26, 2013.] http://xamarin.com/.

82. Xamarin. Building Cross Platform Applications. [Online] 2013. [Cited: 04 26, 2013.] http://docs.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications.

83. Xamarin Developer Center. Architecture. [Online] Xamarin, 2013. [Cited: 09 15, 2013.] http://docs.xamarin.com/guides/android/advanced_topics/architecture.

84. Xamarin Developer Center. Java Integration Overview. [Online] Xamarin, 2013. [Cited: 09 15, 2013.] http://docs.xamarin.com/guides/android/advanced_topics/java_integration_overview.

85. Xamarin. Products. [Online] 2013. [Cited: 04 26, 2013.] http://xamarin.com/products.

86. Xamarin. Xamarin Test Cloud. [Online] 2013. [Cited: 05 10, 2013.] http://xamarin.com/test-cloud.

87. IBM Worklight . Mobile application platform. [Online] 2012. [Cited: 02 04, 2013.] http://www-01.ibm.com/software/mobile-solutions/worklight/.

88. Phillip Redman, John Girard, Monica Basso. Magic Quadrant for Mobile Device. [Online] 05 17, 2012. [Cited: 02 04, 2013.] http://www.notifycorp.com/products/documentation/magic_quadrant_for_mobile_de_230508%20(2).pdf.

89. Improve your mobile application security with IBM Worklight . [Online] 2012. [Cited: 02 04, 2013.] http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14198usen/WSW14198USEN.PDF.

90. IBM Education Assistant. [Online] 10 26, 2012. [Cited: 02 04, 2013.] http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.worklight/plugin_coverpage.html.

91. w3schools. XML Tutorial. [Online] 2013. [Cited: 02 17, 2013.] http://www.w3schools.com/xml/.

92. The Internet Society. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. [Online] 2006. [Cited: 02 17, 2013.] https://tools.ietf.org/html/rfc4510.

93. JSON. Introducing JSON. [Online] 1999. [Cited: 02 17, 2013.] http://www.json.org/.

94. IT Business Edge Network . mash-up. [Online] 2013. [Cited: 02 17, 2013.] http://www.webopedia.com/TERM/M/mash_up.html.

95. IBM Worklight V5.0.5. [Online] 2013. [Cited: 02 04, 2013.] http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14181usen/WSW14181USEN.PDF.

96. IBM Mobile Foundation. IBM Worklight Device Runtime Components. [Online] IBM, 2013. [Cited: 09 14, 2013.] http://www-01.ibm.com/software/mobile-solutions/worklight/components/runtime/.

97. Mobile application development platforms. [Online] 2012. [Cited: 02 04, 2013.] http://www.ibm.com/developerworks/websphere/techjournal/1210_col_amrhein/1210_col_amrhein.html.

98. Jonathan Kempel. IBM Software Group . Top 10 considerations for choosing a mobile. [Online] IBM, 2013. [Cited: 11 10, 2013.] http://www-01.ibm.com/software/solutions/soa/newsletter/june12/top_ten.html.

99. Gomes, Carina. Estudo do Paradigma Computação em Nuvem. Lisboa : Instituto Superior de Engenharia de Lisboa, 2013.

100. João C. Ferreira, Porfírio P. Filipe, Carina Gomes, Gonçalo Cunha, João Silva. Taas – Ticketing as a Service. 2012.

101. Gomes, Carina. Estudo do Paradigma Computação em Nuvem. 2012. Tese de Mestrado no ISEL.

102. Windows Azure. BrokeredMessage. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://www.windowsazure.com/en-us/develop/net/how-to-guides/service-bus-queues/.

103. Windows Azure. Cloud Services. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://www.windowsazure.com/en-us/services/cloud-services/.

104. Windows Azure. Service Bus Topic. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://www.windowsazure.com/en-us/develop/net/how-to-guides/service-bus-topics/.

105. Charles Petzold. ACM Digital Library. Microsoft XNA Framework Edition: Programming Windows Phone 7. [Online] Association for Computing Machinery, 2013. [Cited: 09 21, 2013.] http://dl.acm.org/citation.cfm?id=1965332.

106. Apple. iOS 7 - A nova visão para o sistema operativo móvel. [Online] Apple, 2013. [Cited: 09 21, 2013.] http://www.apple.com/pt/ios/ios7/.

107. MSDN. REST reference. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://msdn.microsoft.com/en-us/library/live/hh243648.aspx.

108. Web Sites. Windows Azure. [Online] Microsoft, 2013. [Cited: 09 03, 2013.] http://www.windowsazure.com/en-us/services/web-sites/.

109. MSDN. BrokeredMessage Class. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.brokeredmessage.aspx.

110. MSDN. Windows Azure Queues and Windows Azure Service Bus Queues - Compared and Contrasted. [Online] Microsoft, 2013. [Cited: 09 21, 2013.] http://msdn.microsoft.com/en-us/library/windowsazure/hh767287.aspx.

111. Interoperability bridges and labs center. Windows Azure SDK for Java. [Online] Microsoft Open Technologies, Inc. , 2013. [Cited: 09 21, 2013.] http://www.interoperabilitybridges.com/projects/windows-azure-sdk-for-java.

112. MSDN. The Autoscaling Application Block. [Online] 2013. [Cited: 03 27, 2013.] http://msdn.microsoft.com/en-us/library/hh680892(v=pandp.50).aspx.

113. MSDN. How to Use Service Bus Queues. [Online] 2013. [Cited: 03 27, 2013.] http://www.windowsazure.com/en-us/develop/net/how-to-guides/service-bus-queues/.

ANEXO A – Definição das regras de validação da mensagem

Definição das regras de validação do objeto *Message* (Figura 23).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:include schemaLocation="Header.xsd"/>
	<xs:element name="Message">
		<xs:annotation>
			<xs:documentation>Structure of input and output
message</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence minOccurs="1" maxOccurs="1">
				<xs:element ref="header" minOccurs="1" maxOccurs="1">
					<xs:annotation>
						<xs:documentation>Header
message</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element ref="bodyMessage" minOccurs="0"
maxOccurs="1"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="bodyMessage">
		<xs:annotation>
			<xs:documentation>Body message</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence minOccurs="1" maxOccurs="1">
				<xs:element name="par" maxOccurs="unbounded">
					<xs:annotation>
						<xs:documentation>Attribute-Value
Pairs</xs:documentation>
					</xs:annotation>
					<xs:complexType>
						<xs:sequence>
							<xs:element name="value"
type="xs:anyType" minOccurs="0">
<xs:annotation>

	<xs:documentation>Attribute value</xs:documentation>
</xs:annotation>
							</xs:element>
						</xs:sequence>
						<xs:attribute name="attribute" type="xs:string"
use="required">
<xs:annotation>
	<xs:documentation>Attribute
name</xs:documentation>
</xs:annotation>
						</xs:attribute>
					</xs:complexType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="header">
```

```
                <xs:annotation>
                        <xs:documentation>Header message</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                        <xs:attribute name="deviceId" type="xs:string" use="required">
                                <xs:annotation>
                                        <xs:documentation>Device
identifier</xs:documentation>
                                </xs:annotation>
                        </xs:attribute>
                        <xs:attribute name="operationId" type="xs:integer" use="required">
                                <xs:annotation>
                                        <xs:documentation>Operation
identifier</xs:documentation>
                                </xs:annotation>
                        </xs:attribute>
                        <xs:attribute name="messageId" type="xs:integer" use="required">
                                <xs:annotation>
                                        <xs:documentation>Message
identifier</xs:documentation>
                                </xs:annotation>
                        </xs:attribute>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

ANEXO B – Definição das regras de validação dos objetos de dados

Definição das regras de validação do objeto de dados *Register* (Figura 24).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:element name="Register">
		<xs:annotation>
			<xs:documentation>Device registration</xs:documentation>
		</xs:annotation>
		<xs:complexType>
			<xs:sequence>
				<xs:element name="Device">
					<xs:annotation>

	<xs:documentation>Device</xs:documentation>
					</xs:annotation>
					<xs:complexType>
						<xs:attribute name="deviceId" type="xs:ID"
use="required">
							<xs:annotation>
								<xs:documentation>Device
identifier</xs:documentation>
							</xs:annotation>
						</xs:attribute>
						<xs:attribute name="operatorId"
use="required">
							<xs:annotation>
								<xs:documentation>Operator
identifier</xs:documentation>
							</xs:annotation>
						</xs:attribute>
						<xs:attribute name="deviceTypeId"
use="required">
							<xs:annotation>
								<xs:documentation>Device
type identifier</xs:documentation>
							</xs:annotation>
						</xs:attribute>
						<xs:attribute name="registrationDate"
use="required">
							<xs:annotation>
								<xs:documentation>Device
registration date</xs:documentation>
							</xs:annotation>
						</xs:attribute>
<xs:attribute name="isOnline" use="required">
							<xs:annotation>
								<xs:documentation>Device
mode</xs:documentation>
							</xs:annotation>
						</xs:attribute>
					</xs:complexType>
				</xs:element>
			</xs:sequence>
		</xs:complexType>
```

```
        </xs:element>
</xs:schema>
```

DEFINIÇÃO DAS REGRAS DE VALIDAÇÃO DO OBJETO DE DADOS *TYPELOGIC* (FIGURA 25).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="TypeLogic">
                <xs:complexType>
                        <xs:sequence>
                                <xs:annotation>
                                        <xs:documentation>Representation of  type logic
object</xs:documentation>
                                </xs:annotation>
                                <xs:element name="deviceTypeId" type="xs:string"
minOccurs="1" maxOccurs="1">
                                        <xs:annotation>
                                                <xs:documentation>Device type
identifier</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="operatorId" type="xs:string" minOccurs="1"
maxOccurs="1">
                                        <xs:annotation>
                                                <xs:documentation>Operator
identifier</xs:documentation>
                                        </xs:annotation>
                                </xs:element>
                                <xs:element name="sourceCode">
                                        <xs:annotation>
                                                <xs:documentation>Source
code</xs:documentation>
                                        </xs:annotation>
                                        <xs:complexType>
                                                <xs:sequence minOccurs="1" maxOccurs="1">
                                                        <xs:element name="execute"
type="xs:string" minOccurs="1" maxOccurs="1">
                                                                <xs:annotation>

        <xs:documentation>Execute operation</xs:documentation>
                                                                </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="method1"
type="xs:string" minOccurs="0" maxOccurs="1">
                                                                <xs:annotation>

        <xs:documentation>Method 1</xs:documentation>
                                                                </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="methodN"
type="xs:string" minOccurs="0" maxOccurs="1">
                                                                <xs:annotation>

        <xs:documentation>Method N</xs:documentation>
                                                                </xs:annotation>
                                                        </xs:element>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

ANEXO C – Definição das regras de validação do protocolo de registo

Formato de resposta, do primeiro pedido (registo) ou de mudança de modo (*online*/*offline*), emitida do adaptador para o dispositivo (Figura 14).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="RegisterResponse">
		<xs:complexType>
			<xs:sequence>
				<xs:annotation>
					<xs:documentation>Representation of register
operation format response</xs:documentation>
				</xs:annotation>
				<xs:element name="resultCode" type="xs:string"
minOccurs="1" maxOccurs="1">
					<xs:annotation>
						<xs:documentation>Register operation result
code</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element name="resultMessage" type="xs:string"
minOccurs="1" maxOccurs="1">
					<xs:annotation>
						<xs:documentation>Register operation result
message</xs:documentation>
					</xs:annotation>
				</xs:element>
				<xs:element name="firstData">
					<xs:annotation>
						<xs:documentation>Data to show in the first
screen</xs:documentation>
					</xs:annotation>
					<xs:complexType>
						<xs:sequence minOccurs="1" maxOccurs="1">
							<xs:element name="screenTitle"
type="xs:string" minOccurs="1" maxOccurs="1">
								<xs:annotation>

	<xs:documentation>First screen title</xs:documentation>
								</xs:annotation>
							</xs:element>
							<xs:element name="resultData">
								<xs:annotation>

	<xs:documentation>Operations data</xs:documentation>
								</xs:annotation>
								<xs:complexType>
									<xs:sequence
minOccurs="1" maxOccurs="1">
<xs:element
name="operationName" type="xs:string" minOccurs="1" maxOccurs="1">

	<xs:annotation>

	<xs:documentation>Operation name</xs:documentation>

	</xs:annotation>
```

```
                                                                    </xs:element>
                                                                    <xs:element
name="operationCode" type="xs:string" minOccurs="1" maxOccurs="1">

        <xs:annotation>

        <xs:documentation>Operation Code</xs:documentation>

        </xs:annotation>
                                                                    </xs:element>
                                                                </xs:sequence>
                                                            </xs:complexType>
                                                        </xs:element>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                            <xs:element name="sourceCode">
                                                <xs:annotation>
                                                    <xs:documentation>Source
code</xs:documentation>
                                                </xs:annotation>
                                                <xs:complexType>
                                                    <xs:sequence minOccurs="1" maxOccurs="1">
                                                        <xs:element name="execute"
type="xs:string" minOccurs="1" maxOccurs="1">
                                                            <xs:annotation>

        <xs:documentation>Execute operation</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="method1"
type="xs:string" minOccurs="0" maxOccurs="1">
                                                            <xs:annotation>

        <xs:documentation>Method 1</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                        <xs:element name="methodN"
type="xs:string" minOccurs="0" maxOccurs="1">
                                                            <xs:annotation>

        <xs:documentation>Method N</xs:documentation>
                                                            </xs:annotation>
                                                        </xs:element>
                                                    </xs:sequence>
                                                </xs:complexType>
                                            </xs:element>
                                        </xs:sequence>
                                    </xs:complexType>
        </xs:element>
</xs:schema>
```