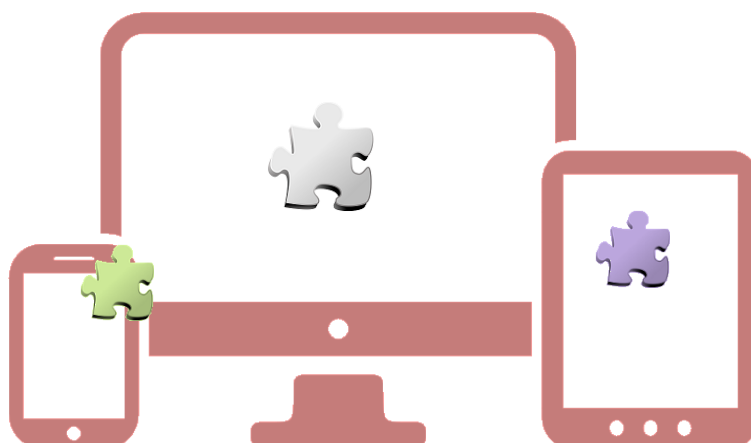




INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Interacção Remota de Objectos em Redes Distribuídas com Dispositivos Móveis

SÉRGIO MIGUEL RODRIGUES

Licenciado

Trabalho de Projecto para obtenção do grau de Mestre em Engenharia na Área de
Especialização em engenharia de electrónica e telecomunicações

Orientador (es):

Doutor, Manuel Martins Barata, Professor coordenador (ISEL)

Júri:

Presidente: Doutor, Mário Pereira Véstias, Professor coordenador (ISEL)

Vogais:

Doutor, Porfírio Pena Filipe, Professor Adjunto (ISEL)

Doutor, Manuel Martins Barata, Professor coordenador (ISEL)

Março de 2016

*Um sistema distribuído é uma colecção de computadores independentes
que aparecem aos utilizadores como um único sistema coerente*

Andrew Tanenbaum

Agradecimentos

Seguramente não irei conseguir agradecer devidamente a todas as pessoas que ao longo do meu Mestrado me ajudaram, directa ou indirectamente a realizar esta etapa da minha formação académica.

Deixo portanto aqui umas curtas palavras que expressam um pouco do que sinto.

Ao Professor Doutor Manuel Barata, expresso o meu profundo agradecimento pela orientação e apoio determinante na elaboração e conclusão deste Trabalho de Projecto.

Aos meus pais um enorme obrigado por acreditarem em mim e naquilo que faço mas principalmente por me apoiarem e não me deixarem desistir.

A ti Claudia, minha companheira de vida e confidente, expresso o meu profundo agradecimento por me aturares e me dares apoio nos meus altos e baixos durante todo este longo processo de conclusão desta etapa.

À Catarina e à Constança um grande agradecimento pois foram as principais responsáveis por eu conseguir desconstrair e conseguir libertar a cabeça quando tudo corria mal.

Espero que agora, após esta etapa, possa compensar todo o carinho, apoio e dedicação que obtive.

A eles, dedico todo este trabalho.

Resumo

Os dispositivos móveis munidos de aplicações informáticas para os mais variados fins, têm vindo a ganhar cada vez mais popularidade. A incorporação de interfaces de comunicação WiFi fez com que esses dispositivos passassem de periféricos multimédia de computadores pessoais, para mais um computador que se liga numa rede local. Desse modo os terminais móveis, cada vez mais, são capazes de executar aplicações bastante complexas devido às suas novas características de *hardware*, com maior capacidade de processamento, memória de trabalho e armazenamento de dados. São, no essencial, um computador compacto munido de tecnologia de telecomunicações, que para além de permitirem as comunicações de dados e voz através das redes de telecomunicações, oferecem ainda um conjunto de funcionalidades de natureza multimédia e localização. Estas características levam a que a concepção de programas para estes dispositivos, seja um dos aspectos fundamentais da investigação e desenvolvimento actual em curso: a computação móvel.

É necessário facilitar o desenvolvimento de aplicações distribuídas que possibilitem que um objecto activo num dispositivo móvel interaja com outros num contexto de objectos distribuídos, independentemente da sua localização.

Este trabalho consiste na proposta e elaboração de uma solução que facilita o desenvolvimento de aplicações distribuídas, libertando o programador da preocupação com os detalhes da comunicação entre as diversas componentes num ambiente de dispositivos móveis.

Palavras-Chave

Android, RPC, RMI, CORBA, XML, JSON, XML-RPC, JSON-RPC, Serviços web.

Abstract

The mobile handheld devices equipped with *software* for various purposes, have been gaining more and more popularity. The incorporation of WiFi communication interfaces made these devices able to pass from multimedia peripherals of personal computers to another computer that connects to a local network. As a consequence, mobile terminals, increasingly, are able to perform fairly complex applications due to their new hardware features, such as: more processing power, bigger working memory size and high capacity data storage. They are essentially a compact computer provided with telecommunications technology, which in addition allows data communications and voice over telecommunications networks, offering yet a range of multimedia features and location. These characteristics led to the design of programs for these devices becoming a fundamental aspect of the current research and development in progress: mobile computing.

It is necessary to facilitate the development of distributed applications that enable an active object in a mobile device to interact with others in the context of distributed objects, regardless of their location.

This work consists in proposing and developing a solution that ease the development of distributed applications, freeing the programmer of the concern about the details of the communication between the various components in a mobile environment.

Keywords

Android, RPC, RMI, CORBA, XML, JSON, XML-RPC, JSON-RPC, Web Services.

Acrónimos

6LoWPAN	– IPv6 over Low power Wireless Personal Area Networks	IPv6 sobre redes de área pessoal sem fios de baixa energia
AAPT	– Android Asset Packaging Tool	Ferramenta de empacotamento de activos do Android
ADB	– Android Debug Bridge	Ponte de depuração do Android
AMQP	– Advanced Message Queueing Protocol	Protocolo avançado de enfileiramento de mensagem
ANR	– Application Not Responding	Aplicação Não está a Responder
API	– Application Programming Interface	Interface de Programação da Aplicação
APK	– Android Package	Pacote Android
ART	– Android Runtime	Ambiente de execução Android
BSD	– Berkeley Software Distribution	Distribuição de software Berkeley
COAP	– Constrained Application Protocol	Protocolo restrito de aplicação
CORBA	– Common Object Request Broker Architecture	Arquitectura de Broker de solicitação de objecto comum
CPU	– Central Processing Unit	Unidade central de processamento
DEX	– Dalvik Executable	Executável Dalvik
DTD	– Document Type Definitions	Documento de Definições de Tipo
DTLS	– Datagram Transport Layer Security	Segurança da camada de transporte de datagrama

ELF	– Extensible Linking Format	Formato extensível de ligamento
HTML	– HyperText Markup Language	Linguagem de marcação de hipertexto
HTTP	– HyperText Transfer Protocol	Protocolo de Transferência de Hipertexto
IDE	– Integrated Development Environment	Ambiente Integrado de Desenvolvimento
IP	– Internet Protocol	Protocolo de Internet
IPC	– Inter-Process Communication	Comunicação entre Processos
JEE	– Java Enterprise Edition	Edição corporativa Java
JMS	– Java Message Service	Serviço de mensagens Java
JSON	– JavaScript Object Notation	Notação de Objecto JavaScript
JVM	– Java Virtual Machine	Máquina Virtual Java
M2M	– Machine-to-Machine	Comunicações de Máquina para Máquina
MQTT	– Message Queue Telemetry Transport	Transporte de telemetria de fila de mensagem
MQTT-SN	– MQTT - Sensor Networks	MQTT para Redes de Sensores
NDK	– Native Development Kit	Kit de desenvolvimento nativo
NFC	– Near Field Communication	Comunicação por campo de proximidade
OEM	– Original Equipment Manufacturer	Fabricante do equipamento original
PAI	– Protocol Agnostic Interface	Protocolo agnóstico de Interface
PDA	– Personal Digital Assistant	Assistente pessoal digital
QoS	– Quality of Service	Qualidade de Serviço
REST	– Representational State Transfer	Transferência de Estado Representacional
RFC	– Request For Comments	Pedido de Comentários

RFID	– R adio- F requency I dentification	Identificação por Radiofrequência
RMI	– R emote M ethod I nvocation	Invocação de método remoto
RPC	– R emote P rocedure C all	Chamada Remota ao Procedimento
RTPS	– R eal T ime P ublish S ubscribe	Publicar e subscrever em tempo real
SCADA	– S upervisory C ontrol A nd D ata A cquisition	Controle de supervisão e aquisição de dados
SCI	– S ystem C all I nterface	Interface de chamada do sistema
SDK	– S oftware D evelopment K it	Kit de desenvolvimento de software
SE	– S tandard E dition	Edição padrão
SGML	– S tandard G eneralized M arkup L anguage	Linguagem Padronizada de Marcação Genérica
SLP	– S amsung L inux P latform	Plataforma Linux da Samsung
SMS	– S hort M essage S ervice	Serviço de mensagens curtas
SOA	– S ervice- O riented A rchitecture	Arquitectura orientada a serviços
SOAP	– S imple O bject A ccess P rotocol	Protocolo Simples de Acesso a Objectos
TCP	– T ransmission C ontrol P rotocol	Protocolo de controlo de transmissão
TCP/IP	– Internet protocol suite TCP/IP	Pilha de protocolos TCP/IP
TSG	– T echnical S teering G roup	Grupo de Coordenação Técnica
UDP	– U ser D atagram P rotocol	Protocolo de datagrama de utilizador
UI	– U ser I nterface	Interface de Utilizador
URI	– U niform R esource I dentifier	Identificador Uniforme de Recursos
URL	– U niform R esource L ocator	Localizador Padrão de Recursos

UTF	– Unicode Transformation Format	Formato de transformação Unicode
W3C	– World Wide Web Consortium	Consórcio para a Rede mundial de computadores
WAN	– Wide Area Network	Rede de longa distância
WS	– Web Service	Serviço <i>Web</i>
WSD	– Web Service Description	Descrição de Serviço <i>Web</i>
WSDL	– Web Service Description Language	Linguagem de Descrição de Serviço <i>Web</i>
XML	– Extensible Markup Language	Linguagem Extensível de Marcação

Índice de Figuras

Figura 1 – Diagrama de Blocos RPC	11
Figura 2 – Arquitectura RMI [13]	12
Figura 3 – Âmbito dos protocolos M2M	17
Figura 4 – Percentagem de utilizadores da Internet entre 2000 e 2013 [47]	24
Figura 5 – Percentagem de detentores de telemóvel de Maio 2012 a Agosto 2014 [48].25	
Figura 6 – Diagrama de blocos da arquitectura Android [53]	29
Figura 7 – Diagrama de blocos do Núcleo do Android	29
Figura 8 – Funcionamento do <i>driver</i> binder [62]	32
Figura 9 – Diagrama de blocos da camada de bibliotecas	33
Figura 10 – Diagrama de blocos do Runtime Android	34
Figura 11 – Diagrama de blocos da <i>Framework</i> Aplicacional	36
Figura 12 – Diagrama de blocos da camada das Aplicações	37
Figura 13 – Processo de instalação de uma aplicação Android [87]	38
Figura 14 – Formato da mensagem MQTT	42
Figura 15 – Formato cabeçalho fixo da mensagem MQTT	42
Figura 16 – Arquitectura SROD	46
Figura 17 – Diagrama de Blocos da <i>Framework</i> SROD	48
Figura 18 – Seriação JSON sem ter em consideração referências cíclicas	53
Figura 19 – Seriação JSON considerando a possibilidade de referências cíclicas	53

Figura 20 – Diagrama de ligações e funções desempenhadas	57
Figura 21 – Ecrã da lista de tarefas	61
Figura 22 – Ecrã da lista de tarefas com duas tarefas adicionadas	63
Figura 23 – Comunicações efectuadas ao invocar o objecto remoto duas vezes	63
Figura 24 – Comunicações entre o servidor e o mediador SROD	64
Figura 25 – Detalhe das comunicações efectuadas ao invocar o objecto remoto	65
Figura 26 – <i>Output</i> apresentado na consola	65
Figura 27 – Ecrã da lista de tarefas com duas tarefas adicionadas	67
Figura 28 – Comunicações RMI	68
Figura 29 – Aplicação Android a correr no emulador	70
Figura 30 – Aplicação Android – Adicionar uma tarefa	71
Figura 31 – Aplicação Android – Tarefa adicionada	73

Índice de Tabelas

Tabela 1 – Fabrico mundial de dispositivos por segmento [49].	26
Tabela 2 – Distribuição mundial de dispositivos por sistema operativo [49]	26
Tabela 3 – Diversas versões Android [55]	30

Índice de Código

Código 1 – Exemplo de uma interface de serviço designada de Calculadora	47
Código 2 – Exemplo da implementação da interface Calculadora	47
Código 3 – Exemplo da criação do proxy dinâmico	48
Código 4 – Exemplo da implementação do InvocationHandler	49
Código 5 – Exemplo de uma referência cíclica.	52
Código 6 – Interface do serviço CourierTasks	61
Código 7 – Implementação do serviço CourierTasksService	62
Código 8 – bloco de código que demonstra a utilização do objecto remoto	62
Código 9 – Instanciação e disponibilização do serviço CourierTasks	62
Código 10 – Acesso e localização do objecto remoto	63
Código 11 – Exemplo de interface remota em RMI	66
Código 12- Exemplo de implementação da interface remota	66
Código 13 – Disponibilização do objecto remoto em RMI	67
Código 14 – Localização e acesso ao objecto remoto em RMI	67
Código 15 – Implementação de uma caixa de diálogo a questionar o utilizador	72

Índice

AGRADECIMENTOS	III
RESUMO.....	V
ABSTRACT	VII
ACRÓNIMOS	IX
ÍNDICE DE FIGURAS	XIII
ÍNDICE DE TABELAS	XV
ÍNDICE DE CÓDIGO	XVII
ÍNDICE.....	XIX
1. INTRODUÇÃO	1
1.1.CONTEXTUALIZAÇÃO E OBJECTIVOS.....	2
1.2.ORGANIZAÇÃO DO DOCUMENTO	3
2. ENQUADRAMENTO	5
2.1.IoT - INTERNET OF THINGS	6
2.2.SOA - SERVICE-ORIENTED ARCHITECTURE.....	6
2.3.SISTEMA DISTRIBUÍDO	7
2.4.ARQUITECTURA DE OBJECTOS DISTRIBUÍDOS	7
2.5.MIDDLEWARE	8
2.6.INVOCAÇÃO REMOTA DE OBJECTOS.....	9
2.7. <i>SOCKETS</i>	10
2.8.RPC - REMOTE PROCEDURE CALL.....	11
2.1.RMI - <i>REMOTE METHOD INVOCATION</i>	12
2.2.SERVIÇOS WEB	13
2.3.CORBA.....	14
2.4.XML-RPC.....	14
2.5.JSON-RPC	15
2.6.HTTP - HYPERTEXT TRANSFER PROTOCOL.....	15

2.7.M2M – COMUNICAÇÕES DE MÁQUINA PARA MÁQUINA	16
2.8.SISTEMAS OPERATIVOS PARA DISPOSITIVOS MÓVEIS	21
2.9.UTILIZAÇÃO DE INTERNET	24
2.10.EQUIPAMENTOS COMPUTACIONAIS	25
3. FRAMEWORKS.....	27
3.1.ANDROID	28
3.2.ARQUITECTURA ANDROID	28
3.3.DESENVOLVIMENTO ANDROID	37
3.4.JSON – JAVASCRIPT OBJECT NOTATION.....	39
3.5.JSON-RPC.....	40
3.6.MQTT	41
4. A FRAMEWORK.....	45
4.1.ARQUITECTURA	46
5. PROVA DE CONCEITO	57
5.1.ESCOLHA DO MEDIADOR MQTT	58
5.2.CONTEXTO.....	60
6. CONCLUSÕES	75
REFERÊNCIAS DOCUMENTAIS.....	77

1. INTRODUÇÃO

Os dispositivos móveis portáteis munidos de aplicações informáticas para os mais variados fins, têm vindo a ganhar cada vez mais popularidade. A incorporação de interfaces de comunicação WiFi fez com que passassem de periféricos multimédia de computadores pessoais, para mais um computador que se liga numa rede local. Desse modo os terminais móveis, cada vez mais, são capazes de executar aplicações bastante complexas devido às suas novas características de *hardware*, com maior capacidade de processamento, memória RAM de trabalho e armazenamento de dados. São, no essencial, computadores compactos munidos de tecnologia de telecomunicações, que para além de permitirem as comunicações de dados e voz através das redes de telecomunicações, oferecem ainda um conjunto de funcionalidades de natureza multimédia e localização.

Considerado o pai da computação ubíqua, Mark Weiser, previu que os computadores tornar-se-iam parte integrante do ambiente e do dia-a-dia do ser humano, ao mesmo tempo que desapareciam dos seus olhos [1]. A computação ubíqua tem como objectivo aumentar a utilização dos computadores, disponibilizando vários computadores por todo o ambiente físico, mas em simultâneo tornando-os efectivamente invisíveis para o utilizador [2]. A portabilidade e mobilidade de muitos desses dispositivos, bem como a sua capacidade de transmitir dados, voz ou vídeo em lugares diferentes através de interfaces de comunicação sem fios caracteriza a computação móvel.

A computação móvel aliada á computação ubíqua aproveita esses pequenos e baratos dispositivos computacionais presentes fisicamente junto do utilizador. Os dispositivos computacionais encontram-se dispersos um pouco por todo o lado auxiliando-nos nas mais variadas tarefas da vida diária, trazem características únicas que têm um forte impacto na concepção e implementação dos novos sistemas informáticos. É necessário compreender todas as implicações de modo a desenvolver e implementar sistemas e infra-estruturas melhores adaptadas a este paradigma, o que leva a que a concepção de programas para estes dispositivos, seja um dos aspectos fundamentais da investigação e desenvolvimento actual relativamente á computação móvel [3].

1.1. CONTEXTUALIZAÇÃO E OBJECTIVOS

Este projecto surgiu do desejo de realizar um trabalho no âmbito dos dispositivos de comunicação móvel, focando-se na necessidade de comunicação entre diversos sistemas. A fragmentação provocada pela diversidade de sistemas operativos móveis e as suas diferentes versões transforma o desenvolvimento de aplicações móveis uma tarefa muito complexa principalmente quando se tem em consideração as escolhas dos diversos protocolos de comunicação para tentar uma maior compatibilidade dentro do ecossistema. Existem protocolos proprietários e abertos, bem como protocolos intrínsecos á linguagem ou plataforma de desenvolvimento da aplicação para determinado equipamento ou sistema operativo. Serão analisados mecanismos de invocação remota de objectos e a sua aplicabilidade para dispositivos móveis. Serão analisados protocolos existentes, tendo em vista a linguagem, o paradigma de programação, a licença de uso, entre outros, de forma a se poder ter um conjunto de indicadores para uma decisão relativamente ao uso de determinada *framework*

O objectivo deste trabalho incide no desenvolvimento de uma *framework* de comunicação que seja transparente para o utilizador, tendo como ponto de referência uma aplicação móvel capaz de responder as várias particularidades do ambiente multiplataforma e multiequipamento. De seguida vai-se avaliar se o desempenho da *framework* nessa aplicação tem impacto sobre a sua usabilidade como forma de validar o conceito e a aplicabilidade da *framework* num contexto prático, já que a crescente introdução de dispositivos móveis como nós de um sistema colaborativo, acompanha a crescente necessidade de uma pensada escolha de *framework* de comunicação homogénea aquando a fase de desenho do próprio sistema.

1.2. ORGANIZAÇÃO DO DOCUMENTO

Este documento está organizado da seguinte forma:

- No 1º capítulo estão enumeradas as motivações, o contexto, o problema e os objectivos do projecto.
- No 2º capítulo é enquadrado o sistema, tendo em conta o foco deste projecto assim como o trabalho relacionado nesta área de estudo.
- No 3º capítulo é apresentado o funcionamento das *frameworks* identificadas.
- No 4º capítulo é apresentada a solução desenvolvida.
- No 5º capítulo é apresentado o ambiente de testes bem como os testes efectuados.
- No 6º capítulo são apresentadas as considerações finais, onde é efectuado um balanço sobre o projecto realizado, com especial atenção nos objectivos propostos. Neste capítulo são apresentadas algumas ideias relativas a trabalho futuro.

2. ENQUADRAMENTO

Mark Weiser em 1988, no seu principal artigo, denominado *The Computer for the 21st Century* (publicado em 1991), criou um conceito interessante, segmentou a computação em três eras e designou de computação ubíqua a terceira dessas eras.

Na primeira era da computação, apareceram os *mainframes* (computadores de grande porte), partilhados por várias pessoas. A segunda seria a era dos computadores pessoais onde cada pessoa utiliza o seu próprio computador. A terceira, a era da computação ubíqua, a era da tecnologia “calma”, onde a tecnologia recua para o plano de fundo das nossas vidas, onde habitam os dispositivos de tamanho reduzido, permitindo às pessoas a utilização de vários dispositivos que se integram à nossa vida diária, até serem completamente indistinguíveis [4].

Estamos a viver actualmente o nascimento de uma nova fase na Internet, a Internet das coisas. Nessa fase, a rede passa a interligar vários tipos de objectos e dispositivos inteligentes, que vão interagir entre si e connosco, por forma a tornar o nosso dia-a-dia mais fácil. A Internet e os computadores tradicionais estão a desaparecer, pois estão cada vez mais presentes em tudo, que já nem reparamos neles, esperamos simplesmente que estejam lá, para os utilizar sem muito esforço.

2.1. IoT - INTERNET OF THINGS

A *Internet of Things* (IoT) ou Internet das Coisas é uma tecnologia que tem como base a interligação dos objectos do cotidiano, permitindo um ecossistema de aplicações e serviços inteligentes, cujo objectivo é melhorar e simplificar a vida das pessoas. A Internet das Coisas é um mundo onde os objectos físicos estão perfeitamente integrados na rede de informação e onde se podem tornar participantes activos nos processos de negócio. Os serviços estão disponíveis para interagir através da Internet com esses “objectos inteligentes”, consultar o seu estado e qualquer informação que lhes está associada, tendo em consideração aspectos de segurança e privacidade [5].

A primeira fase da Internet das Coisas refere-se ao aparecimento de códigos de barras e à identificação por radiofrequência (RFID), como forma de ajudar a automatizar o inventário, *rastreamento* e à identificação básica. A segunda fase prevê uma ligação forte entre sensores, objectos, dispositivos, dados e aplicações. Esta segunda fase é vista como sendo uma lógica de negócio distribuída em grande escala, onde a Service-Oriented Architecture (SOA) é reconhecida, desde há vários anos, como sendo o estilo arquitectural mais adequado.

2.2. SOA - SERVICE-ORIENTED ARCHITECTURE

A SOA pode ser definida como um paradigma para organizar e utilizar recursos distribuídos, que podem estar sob o controle de diferentes domínios proprietários, fornecendo um meio uniforme para oferecer, descobrir, interagir e de utilizar as capacidades desses recursos para produzir o efeito desejado, consistentes com as suas pré-condições e obter expectativas mensuráveis [6].

De acordo com este modelo, numa das possíveis interacções entre os principais componentes de um SOA básico, um fornecedor de serviços pública a sua interface de serviço através de um serviço de registro, onde um utente (invocador ou consumidor) o poderá encontrar e posteriormente ligar-se a esse fornecedor de serviço [6].

O conceito central do modelo de referência SOA é a existência de serviços que fornecem acesso a recursos por interfaces bem definidas a serem exercidas na sequência de um contracto de serviços com restrições e políticas [7].

Isso permite um acoplamento fraco dos serviços (minimizando dependências mútuas) que é um dos principais requisitos para se construir *software* orientado a objectos de qualidade, já que o acoplamento fraco mede o quanto uma classe, depende de, ou está relacionada a outra classe ou subsistema.

Os serviços são prestados pela entidade “prestador de serviços”, e estão a ser utilizados por uma segunda entidade, os “consumidores de serviços”. Os serviços podem ser compostos com base noutros serviços existentes, aderindo, assim, ao princípio da reutilização. Eles devem ser autónomos no sentido de que devem apenas controlar a lógica que encapsulam, são descritos e recuperáveis publicamente através de mecanismos de descoberta [7].

O SOA é uma abordagem para a construção de sistemas distribuídos que fornecem funcionalidades de aplicações em forma de serviços para aplicações do utilizador final ou para construir outros serviços.

2.3. SISTEMA DISTRIBUÍDO

Um sistema distribuído pode ser caracterizado através de um modelo computacional de cliente-servidor ou *peer-to-peer*. Coulouris refere que o modelo computacional cliente-servidor é representado por um cliente que realiza requisições para o servidor e este retorna ao cliente o resultado do processamento, é de notar que um servidor também pode ser cliente de outro servidor. O modelo *peer-to-peer* caracteriza-se pela descentralização das funções na rede, onde cada nó desempenha as mesmas funções e não existem nós dedicados à prestação de serviços já que cada nó realiza tanto funções de servidor quanto de cliente. No caso de N nós, cada um pode interagir com qualquer outro, sendo o padrão de comunicação dependente dos requisitos da aplicação [8].

2.4. ARQUITECTURA DE OBJECTOS DISTRIBUÍDOS

Aplicações de objectos distribuídos são compostas por objectos, unidades individuais de blocos de *software* em execução que combina funcionalidades e dados, que frequentemente (mas nem sempre) representam algo do mundo real.

Tipicamente existem múltiplas instâncias de objectos de um determinado tipo. Para cada tipo de objecto define-se uma interface. A interface é o contracto que o servidor oferece

aos clientes que o invocam. Qualquer cliente que queira invocar uma operação no objecto deve utilizar a sua interface para especificar a operação que quer efectuar e para empacotar os argumentos que envia. Quando a invocação chega ao objecto de destino, a mesma definição de interface é utilizada para desempacotar os argumentos para que o objecto possa efectuar a operação pedida com eles [9].

Uma das principais características de um sistema distribuído orientado a objectos é permitir a operação com objectos remotos. Através de uma aplicação cliente orientada a objectos, podemos obter uma referência para um objecto que oferece o serviço desejado e, através dessa referência, invocar métodos desse objecto, mesmo que a instância desse objecto esteja numa máquina diferente daquela do objecto cliente. O funcionamento por objectos distribuídos pode ser caracterizado de uma forma semelhante ao modelo *peer-to-peer*, porém utiliza-se um *middleware* como intermediário no processo de comunicação.

2.5. MIDDLEWARE

O *Middleware* é a camada de *software* que proporciona um modelo de programação, mascarando a heterogeneidade do *Hardware*, Sistemas Operativos e Linguagens de Programação, garantindo assim a interoperabilidade entre os diversos blocos de *software*. A interoperabilidade é a capacidade de dois ou mais sistemas (computadores, meios de comunicação, redes, *software* e outros componentes de tecnologia da informação) de interagir e de trocar dados de acordo com um método definido, de forma a obter os resultados esperados. Ou seja, normalmente disponibiliza um conjunto de blocos (*building blocks*) que facilitam o desenvolvimento de sistemas distribuídos [8].

De acordo com Coulouris, “Um sistema distribuído é um conjunto de blocos de *hardware* ou *software*, localizados em computadores de uma rede, que comunicam e coordenam as suas acções unicamente através de mensagens”.

Um sistema distribuído orientado a objectos prevê os mecanismos necessários para uma gestão transparente da comunicação, de modo a que o programador se ocupe com a lógica da aplicação. Sistemas distribuídos também exigem que a computação em execução nos diferentes espaços de endereçamento, potencialmente em diferentes máquinas, seja capaz de interagir [9].

2.6. INVOCACÃO REMOTA DE OBJECTOS

A invocação remota de objectos refere-se a programas que efectuem chamadas a objectos noutra espaço de memória, possivelmente noutra máquina. Nada se conhece relativamente ao objecto receptor da chamada, sem ser que suporta uma determinada interface. Por exemplo, no caso de um cliente de um objecto distribuído, não deve saber a linguagem de implementação, nem a arquitectura de *hardware*. Para o cliente, o serviço ou objecto remoto deve aparentar estar no mesmo sistema, e tudo o que precisa de saber é o nome (ou a localização) do objecto ou serviço remoto. A invocação remota tem várias vertentes [10]:

- **Procedimental vs. Orientada a objectos** – Na programação procedimental, o código está organizado em pequenos “procedimentos” que usam e alteram os dados, enquanto na programação orientada a objectos, os dados e as funções relacionadas são reunidas num “objecto”. Os dados de um objecto só devem ser manipulados por funções do objecto.
- **Protocolos proprietários vs. Não proprietários** – Um protocolo proprietário é propriedade de uma organização ou indivíduo, que pode impor restrições e manter a especificação em segredo de modo ter o controlo do protocolo. Um protocolo não proprietário tem a sua especificação pública e está disponível para ser utilizado pelo público em geral sem restrições.
- **Protocolos especializados vs. “Em cima do HTTP”** – O HTTP é um protocolo simples de pergunta/resposta que pode ser utilizado como transporte a um protocolo de invocação remota, mas pode não ser possível a sua utilização ou não ter as características necessárias a determinado ambiente e ser utilizado um protocolo de transporte especializado.
- **Binário vs. Baseado em texto** – Os protocolos binários são mais compactos e rápidos de processar, mas os protocolos baseados em texto são melhores em termos de legibilidade, facilidade de reimplementação e de depuração.
- **Linguagem única vs. Agnóstico em termos de linguagem** – Um protocolo baseado numa única linguagem tira partido de todas as características dessa linguagem enquanto um protocolo agnóstico em termos de linguagem tem de fornecer uma interface comum para a definição semântica aplicável de modo a mitigar o risco de que a ligação de uma determinada linguagem possa reduzir a compatibilidade com outras linguagens.

- **Plataforma única vs. Agnóstico em termos de plataforma** – Um protocolo que apenas está previsto para uma plataforma pode retirar todo o partido das características da plataforma sobre a qual está a executar, enquanto um protocolo agnóstico em termos de plataforma, tem de funcionar igualmente em mais de uma plataforma.

2.7. *SOCKETS*

Actualmente a maioria da comunicação entre computadores baseia-se no protocolo de Internet (IP). A API de *sockets* é uma interface de programação fornecida pelo sistema operativo, que permite aos programas controlar e utilizar *sockets* de rede.

Num modelo orientado à ligação cliente-servidor, o *socket* tem um fluxo típico de eventos. No processo servidor aguarda pedidos de um cliente para a prestação de um dos seus serviços. A troca de dados entre o cliente e o servidor ocorre quando um cliente se liga ao servidor e envia-lhe um pedido onde especifica o nome do serviço pretendido e os dados necessários para a sua execução. O servidor executa o pedido do cliente e envia a resposta de volta para o cliente [11].

Para um mecanismo de comunicação básico, as linguagens de programação suportam a utilização de *sockets*, que são flexíveis o suficiente para a comunicação em geral. No entanto, a utilização de *sockets* obriga a que tanto o cliente e o servidor implementem protocolos a nível aplicacional para codificar e decodificar a troca de mensagens, e o desenho de tais protocolos pode ser complexo e propenso a erros.

Uma alternativa à utilização de *sockets* é o Remote Procedure Call (RPC), que abstrai a interface de comunicação ao nível da chamada de procedimento. Em vez de ter de trabalhar directamente com os *sockets*, o programador tem a ilusão de chamar um procedimento local, quando na verdade os argumentos da chamada são empacotados e enviados para o destino remoto da chamada.

Esta ideia de abstrair a parte de comunicação foi introduzida pela primeira vez com RPC, mas o RPC não segue o paradigma de orientação a objectos.

2.8. RPC - REMOTE PROCEDURE CALL

O modelo de execução da Chamada de Procedimento Remoto é similar ao modelo de execução da chamada de procedimento local, no qual a rotina que invoca o procedimento coloca os argumentos numa área de memória bem conhecida e transfere o controlo para o procedimento a executar. O procedimento lê os argumentos e processa-os. Eventualmente a rotina retoma o controlo do fluxo, extraindo o resultado da execução do procedimento de uma área bem conhecida da memória. Após isso, a rotina prossegue com a execução normal. O bloco de código responsável pela conversão (*Marshalling*) dos parâmetros utilizados numa chamada RPC e pela desconversão (*Unmarshalling*) dos resultados passados do servidor após a sua execução designa-se por *stub*, já do lado do servidor o responsável pelo processo inverso designa-se por *skeleton* [12]. A figura 1 mostra um diagrama de blocos demonstrativo do fluxo de funcionamento do RCP.

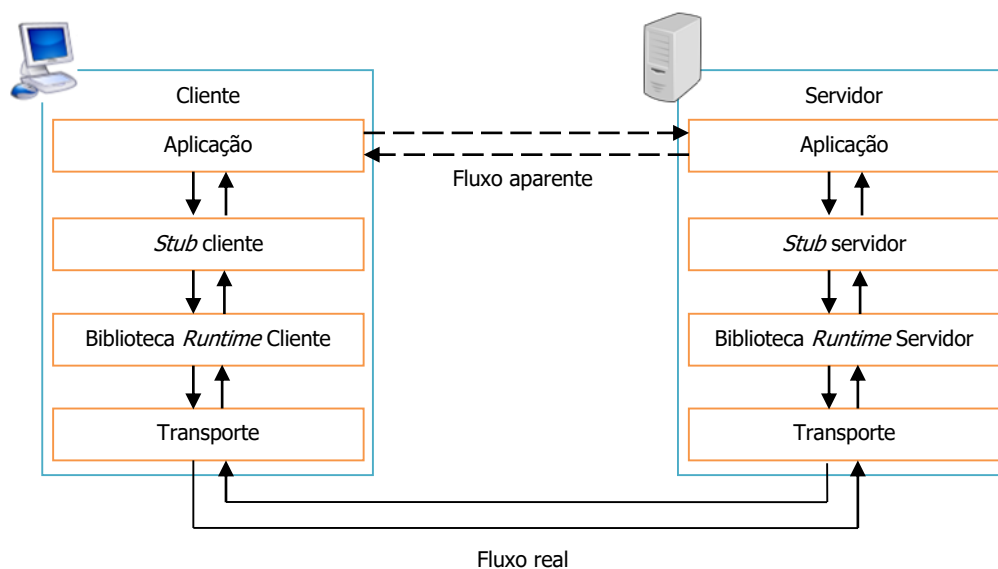


Figura 1 – Diagrama de Blocos RPC

O RPC é aplicado no caso da programação procedimental, nas linguagens de programação orientadas a objectos e nos sistemas de objectos distribuídos, onde é necessária a comunicação entre objectos do mesmo programa que residem em diferentes espaços de endereçamento, é necessário recorrer a outro paradigma. Este novo paradigma terá de fornecer a mesma semântica da invocação a objectos, e permitir que os sistemas de objectos distribuídos consigam invocar os métodos de objectos remotos.

2.1. RMI - *REMOTE METHOD INVOCATION*

Do ponto de vista do programador, a invocação remota terá de ser igual e alinhada com o paradigma de programação que se está a utilizar. O RMI é a adaptação do modelo RPC num modelo que disponibiliza as mesmas funcionalidades num paradigma orientado a objectos específicos da plataforma Java. Num modelo orientado a objectos não existem procedimentos ou funções, só irão existir métodos que operam nos objectos.

Essencialmente no RMI um cliente chama um método num um objecto (remoto) numa JVM (*Java Virtual Machine*) diferente (que poderá estar ou não numa máquina diferente). O cliente não necessita de conhecer nenhum tipo de detalhes sobre o protocolo de rede (portos ou *sockets*), já que o RMI fornece os mecanismos para que a comunicação entre cliente e servidor e a troca de informação seja possível [13].

Uma aplicação RMI costuma consistir em dois programas separados, um cliente e um servidor. Um servidor típico instancia objectos remotos, torna as referências para esses objectos acessíveis e espera por clientes que invoquem os métodos desses objectos. Um cliente típico obtém uma referência remota para um ou mais objectos remotos de um servidor e invoca um ou mais métodos. A figura 2 demonstra uma aplicação distribuída RMI que utiliza o registo RMI para obter uma referência a um objecto remoto. A Figura 2 também mostra que o sistema RMI utiliza um servidor Web existente para transferir as definições de classe, do servidor para o cliente e do cliente para o servidor, para objectos quando necessário..

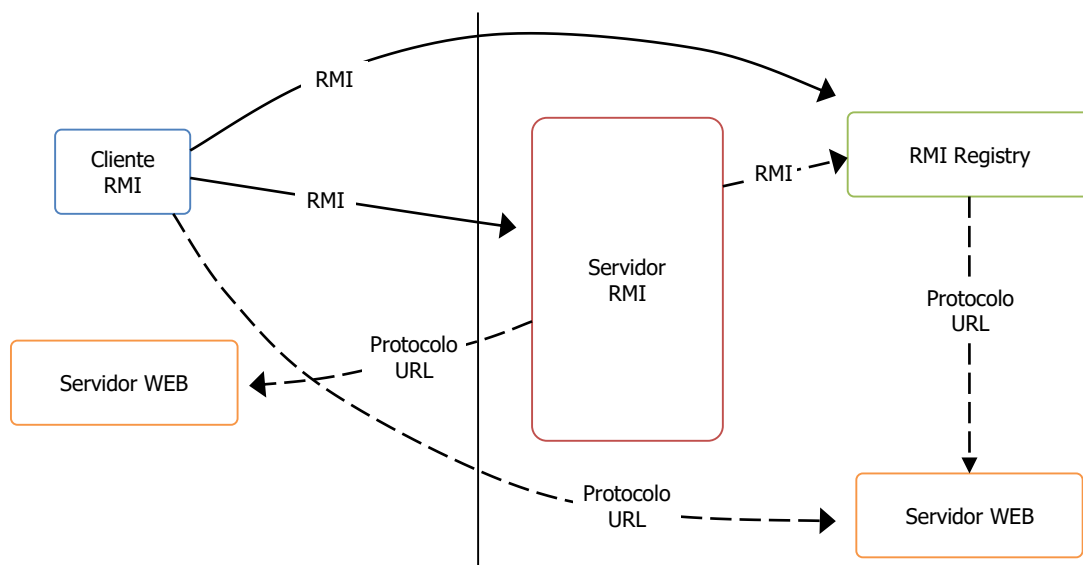


Figura 2 – Arquitectura RMI [13]

Este tipo de aplicações é algumas vezes designado de Aplicação de Objectos Distribuídos, e na plataforma Java este paradigma disponibiliza as seguintes funcionalidades [13]:

- **Localizar Objectos Remotos** - Uma aplicação pode usar vários mecanismos para obter referências de objectos remotos. Por exemplo, uma aplicação pode registrar o objecto remoto com a ferramenta de nomes do RMI (“rmiregistry”), ou pode passar e retornar referências aos objectos remotos como parte de sua operação normal.
- **Comunicar com Objectos Remotos** - Os detalhes de comunicação entre os objectos remotos são tratados pelo RMI, ou seja, para o programador, a comunicação remota é semelhante a uma chamada a um método local.
- **Descarregar “bytecodes” de objectos móveis** - Como o RMI permite que objectos remotos sejam passados como parâmetros numa função, e fornece os mecanismos necessários para descarregar o código dos objectos remotos caso a classe desses objectos não esteja definida na máquina virtual do receptor. Os tipos e o comportamento de um objecto, apenas disponíveis numa máquina virtual, podem portanto, ser transmitidos para outra máquina virtual, possivelmente remota.

A chamada de métodos remotos da plataforma Java foi especificamente projectada para executar no ambiente homogéneo resultante da existência de uma máquina virtual Java em todos os nós do sistema, portanto sempre que possível, o sistema deve tirar proveito do modelo de objectos e das funcionalidades da plataforma Java.

2.2. SERVIÇOS WEB

Os Serviços Web do W3C e o SOAP (Simple Object Access Protocol) são independentes de linguagem, de plataforma e do modelo de objecto, e assim o exigem já que as descrições das interfaces escritas em WSDL são de uma elevada complexidade. O SOAP utiliza XML para ser legível por humanos e como protocolo da camada applicacional especifica o HTTP [14] (mas permite que outros protocolos possam ser utilizados desde que estejam em conformidade com as especificações originais). O HTTP ganhou uma maior aceitação já que funciona bem com infra-estrutura actual de internet e resolve os potenciais problemas da utilização de *firewalls*.

Os Serviços Web não guardam estado, onde cada pedido resulta na criação de um novo objecto para atender à solicitação. Os serviços Web nada sabem sobre o cliente que está a fazer a solicitação, comunicando através de mensagens num formato específico conhecido como o SOAP [15].

Os Serviços Web serializam objectos através do XML que vai contido nas mensagens SOAP e só pode tratar itens que possam ser totalmente expressos em XML.

2.3. CORBA

O CORBA (Common Object Request Broker Architecture) é a arquitectura padrão criada pelo Object Management Group para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogéneos. O CORBA actua de modo a que os objectos possam comunicar de forma transparente ao utilizador, mesmo que para isso seja necessário interoperar com outro *software*, noutro sistema operativo ou noutra ferramenta de desenvolvimento. CORBA é um dos modelos mais populares de objectos distribuídos [16].

2.4. XML-RPC

O XML-RPC é um protocolo de chamada de procedimento remoto que utiliza XML para codificar as suas chamadas e HTTP como o mecanismo de transporte. É um protocolo simples, definido com poucas linhas de código em oposição com a maioria dos sistemas de RPC. O cliente neste caso normalmente é um bloco de *software* que quer chamar um único método dum sistema remoto. Podem ser passados vários parâmetros de entrada para o método remoto, mas apenas é retornado um único valor. Os tipos de parâmetros permitem o aninhamento de parâmetros em mapas e listas, assim estruturas maiores podem ser transportadas. O XML-RPC pode ser usado para transportar objectos ou estruturas como parâmetros de entrada e saída [17].

O XML-RPC é mais simples de perceber e utilizar do que o SOAP pois:

- Permite apenas um método de serialização de método, enquanto o SOAP define várias codificações diferentes;
- Tem um modelo de segurança mais simples;

- Não exige (nem suporta) a criação de descrições de serviço WSDL, apesar de o XRDl fornecer um subconjunto simples das funcionalidades fornecida pelo WSDL.

2.5. JSON-RPC

JSON-RPC é um protocolo de chamada de procedimento remoto codificado em JSON. É um protocolo muito similar ao XML-RPC, que define apenas uns poucos tipos de dados e comandos. Em comparação com o XML-RPC ou com o SOAP, este protocolo permite a comunicação bidireccional dentro do mesmo canal de comunicação onde cada uma das pontas (cliente ou servidor) pode chamar ou enviar notificações para a outra. Permite também o envio de múltiplas chamadas de uma ponta para a outra autorizando até que sejam respondidas fora de ordem [18].

Uma comunicação JSON não está obrigada a utilizar o HTTP para transporte, pelo que também podem ser usados *sockets* TCP/IP para criar aplicações web mais responsivas com o JSON-RPC.

2.6. HTTP - HYPERTEXT TRANSFER PROTOCOL

HTTP (HyperText Transfer Protocol, ou protocolo de transferência de hipertexto) é um dos pilares da World Wide Web, tendo sido desenvolvidos em conjunto com o HTML (HyperText Markup Language) e o URL (Uniform Resource Locator) no final dos anos 80. A primeira versão do protocolo designado de HTTP/1.0 está descrito no RFC1945, enquanto a versão usada actualmente, HTTP/1.1 encontra-se descrita no RFC2616, estando a evolução do protocolo a ser coordenada pelo W3C.

O HTTP é basicamente um protocolo cliente-servidor, do tipo solicitação/resposta onde um cliente faz uma solicitação a um servidor, que geralmente está á escuta no porto 80 através dum pacote HTTP, esse servidor responde com outro pacote HTTP que contém o recurso solicitado. O protocolo HTTP depende do protocolo TCP para comunicações. Uma das principais características do protocolo é ser completamente sem estado (*stateless*): o servidor não necessita de manter as informações de estado em cada tentativa de comunicação [19].

Os modelos simples de solicitação/resposta perguntam “o saldo da conta bancária mudou?” e a resposta obtida é “não, o saldo não mudou”. Então passado alguns minutos verifica novamente e obtém a mesma resposta. Pode parecer um problema insignificante, mas na verdade é um problema muito real. Imaginando que vários clientes verificam obsessivamente o seu saldo ao longo do dia, irão infligir uma carga adicional ao servidor de *backend*.

A comunicação M2M já começou a ganhar o seu espaço devido à redução de energia e de tempo necessário para a informação ser recolhida e transmitida entre máquinas.

2.7. M2M – COMUNICAÇÕES DE MÁQUINA PARA MÁQUINA

O termo comunicações de máquina para máquina (M2M) não se refere a uma ou várias tecnologias específicas de rede nem de comunicação, refere-se sim a tecnologias que permitem aos sistemas com e sem fios comunicar com outros dispositivos do mesmo tipo.

A comunicação M2M moderna evoluiu para além da ligação de um-para-um e transformou-se num sistema de redes que transmitem dados para aparelhos pessoais. Uma série de tecnologias-chave de troca de mensagens estão a surgir para apoiar a próxima geração de aplicações da Internet das coisas. Um dispositivo M2M pode ligar-se ao servidor M2M directamente através de uma ligação para uma rede WAN (por exemplo, rede móvel 3G / 4G) ou um através de um ponto de agregação M2M que agrega e processa os dados de diversos dispositivos M2M. Existem vários protocolos de rádio de baixo custo tais como IEEE 802.11, IEEE802.15 e comunicações via rede eléctrica que estes dispositivos podem utilizar para comunicar [20].

O âmbito das comunicações, demonstrado na figura 3, deve ser um dos requisitos a considerar aquando a escolha de uma das várias tecnologias de mensagens M2M, tais como o DDS, COAP, MQTT, AMQP, JMS e REST via HTTP. Cada um deles pode ser utilizado para ligar dispositivos numa rede distribuída. No entanto deve ser considerada a sua capacidade de suportar os diferentes cenários operacionais, tais como a comunicação entre dispositivos e a comunicação para a nuvem. Deve-se também ter em consideração os principais requisitos do sistema, tais como o desempenho, a interoperabilidade, a tolerância a falhas, a qualidade de serviço e a segurança, algumas aplicações M2M vão necessitar de conectividade entre dispositivos, os protocolos que

funcionem sobre as ligações Peer-to-peer podem ser uma das respostas, dependendo dos requisitos de latência e do tipo de informações trocadas [21].

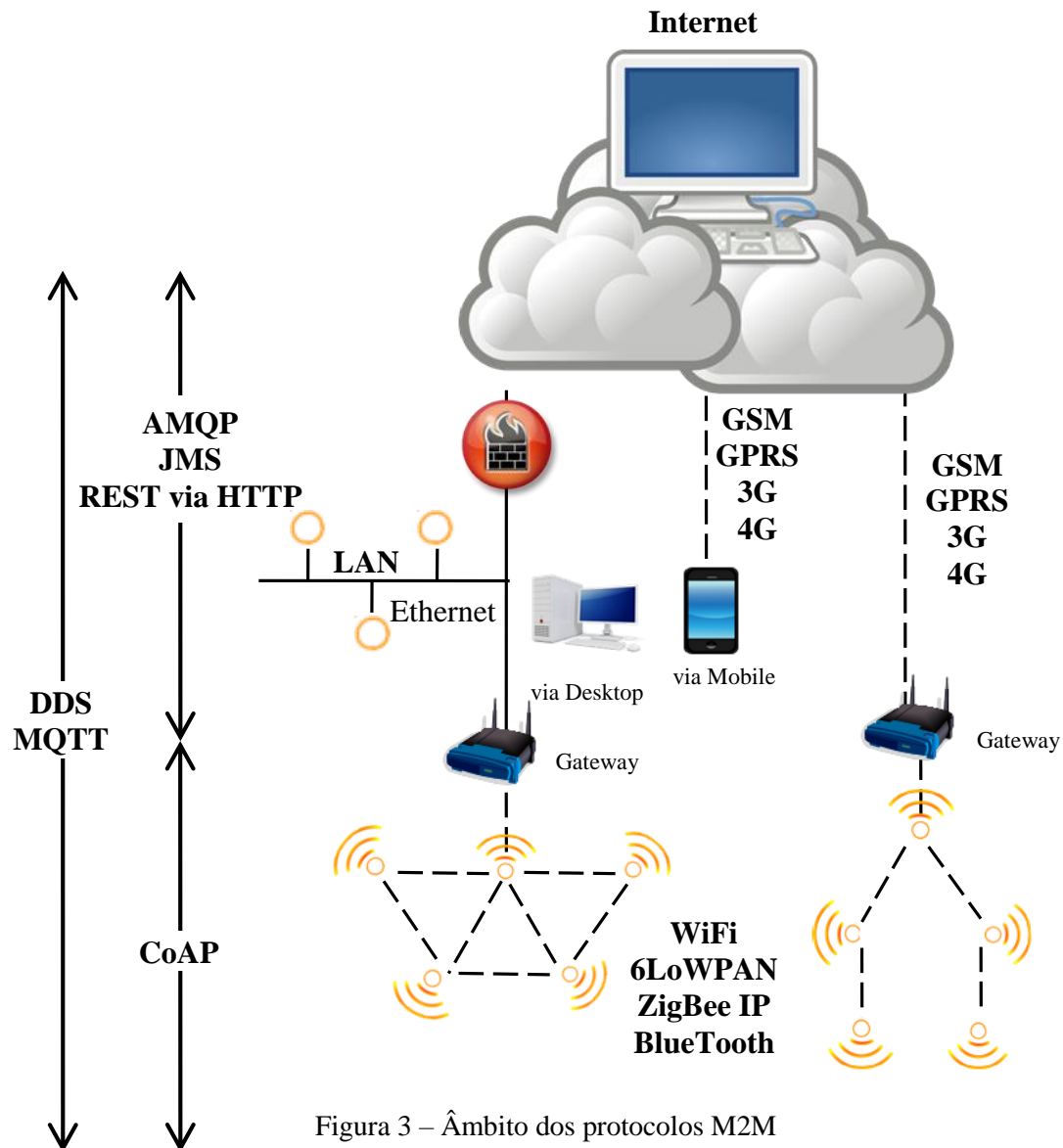


Figura 3 – Âmbito dos protocolos M2M

2.7.1. JMS - JAVA MESSAGE SERVICE

O JMS é uma API Java para o envio de mensagens entre dois ou mais clientes. JMS é parte da plataforma Java Enterprise Edition (JEE), sendo definido pela especificação JSR914. É um padrão de mensagens que permite que os componentes de aplicações baseadas em JEE possam criar, enviar, receber e ler mensagens. Para utilizar o JMS, é preciso ter um fornecedor de JMS que pode gerir as sessões, filas e tópicos. A partir da versão 1.4 do JEE, um fornecedor de JMS está embutido em todos os servidores de aplicações Java EE [22].

2.7.2. AMQP - ADVANCED MESSAGE QUEUEING PROTOCOL

O AMQP é um protocolo aberto de camada aplicacional para middleware orientado a mensagens. O AMQP é orientado à mensagem, tem roteamento (incluindo ponto-a-ponto e de publicação e subscrição), confiabilidade e segurança. O AMQP indica o comportamento do fornecedor de mensagens e do cliente, na medida em que as implementações de diferentes fornecedores são interoperáveis. O AMQP descreve o formato dos dados que são enviados através da rede, desta forma qualquer ferramenta pode criar e interpretar as mensagens, independentemente da sua linguagem de implementação. A versão actual da especificação do protocolo AMQP é a versão 1.0 e centra-se em características essenciais, que são necessárias para a interoperabilidade na escala Internet. É menos explícita em termos de roteamento do que as versões anteriores, já que são as primeiras funcionalidades a ser rigorosamente padronizadas [23].

2.7.3. MQTT- MESSAGE QUEUE TELEMETRY TRANSPORT

O Message Queue Telemetry Transport (MQTT) é um protocolo leve e aberto de publicação/subscrição que funciona através de TCP/IP. O MQTT foi desenvolvido pela IBM para ser utilizado na comunicação com sensores utilizando uma baixa largura de banda e comunicação não confiável. Foi adaptado para redes de sensores sem fio através da especificação MQTT-SN [24].

O MQTT foi desenhado originalmente para fornecer um *middleware* aberto para dados em sistemas SCADA (Supervisory Control And Data Acquisition), com vista a possibilitar e simplificar a ligação dos sistemas de telemetria com outras aplicações empresariais de mais alto nível. O MQTT é mais eficiente do que as técnicas actuais de *pooling*, pelo que se torna menos exigente da rede e do desempenho de CPU, tornando-o assim mais escalável. No entanto, apresenta segurança limitada. Na versão 3.1 do protocolo é possível indicar no pacote MQTT um nome de utilizador e a sua respectiva senha [25].

A Criptografia através da rede pode ser implementada utilizando o SSL, que é independentemente do protocolo MQTT, mas isso trará uma sobrecarga adicional à rede, já que o protocolo SSL não é o mais leve dos protocolos. Pode ser adicionada segurança adicional se as aplicações criptografarem os dados que enviam e recebem, mas isso não é algo definido no protocolo de modo a mantê-lo simples e leve.

2.7.4. REST VIA HTTP - REPRESENTATIONAL STATE TRANSFER

O Representational State Transfer (REST) é um tipo de arquitectura que consiste num conjunto de restrições arquitectónicas aplicadas a componentes, conectores e elementos de dados. O REST ignora os detalhes da implementação do componente e sintaxe do protocolo de modo a concentrar-se nos papéis de componentes, as restrições sobre a sua interacção com os outros componentes, e com a sua interpretação dos elementos de dados [26]. O REST fornece um modo simples de comunicação cliente-servidor, que é útil para sistemas que precisam de comunicar através da Internet, mas não disponibiliza trocas de mensagens assíncronas. É um modelo sem estado apoiado pelo HTTP, o que pode simplificar a implementação do servidor, mas pode ser necessário incluir informação adicional em cada pedido que terá de ser interpretada pelo servidor. Isto pode ser muito ineficiente em termos de tempo de solicitação e de processamento, bem como de recursos consumidos (por exemplo, número de ligações (TCP/IP) [26].

2.7.5. COAP – CONSTRAINED APPLICATION PROTOCOL

O COAP é um protocolo de nível de aplicação (nível 7) no modelo OSI que é adequado para pequenos dispositivos com recursos limitados, permitindo-lhes comunicar interactivamente através de rede. COAP é ideal para pequenos sensores de baixa potência, interruptores e aparelhos semelhantes que têm de ser controlados através da rede de Internet. O COAP foi projectado para ser um protocolo RESTful que permite tanto a comunicação síncrona e assíncrona. O protocolo não foi projectado para ser um substituto para HTTP nem para estar profundamente separado da Web. Um dispositivo COAP está ligado a um sistema baseado em nuvem através de um *proxy* HTTP, utilizando um mapeamento COAP-HTTP que irá provocar uma sobrecarga adicional de comunicação e um aumento da latência de mensagens. COAP consiste em dois tipos de mensagens: as solicitações e as respostas. A especificação do COAP obriga o uso de UDP, e opcionalmente o DTLS (Datagram Transport Layer Security), garantindo um alto nível de segurança nas comunicações. O comprimento do corpo da mensagem é definido pelo comprimento do datagrama, já que quando utilizado o UDP toda a mensagem deve caber dentro de um único datagrama e quando é utilizado com 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks.), o RFC4944 define que a mensagem deve caber numa única trama IEEE802.15.4 com vista a minimizar a fragmentação [27].

2.7.6. DDS – DATA DISTRIBUTION SERVICE

O DDS é o padrão de middleware M2M do Object Management Group (OMG) que visa permitir escalabilidade em tempo real, confiabilidade, alto desempenho e trocas de dados entre publicadores e subscritores por forma a atender às necessidades das aplicações tais como o mercado financeiro, controlo de tráfego aéreo, a gestão de redes inteligentes. A norma DDS contém uma API bem definida e fácil de usar, permitindo aos programadores a escrita de código portátil que irá funcionar com qualquer aplicação compatível com DDS. O padrão DDS faz referência ao protocolo Real Time Publish Subscribe (RTPS) que define o formato dos dados que são transmitidos, permitindo que as aplicações criadas com diferentes implementações DDS possam comunicar, ou interagir entre elas [28].

2.8. SISTEMAS OPERATIVOS PARA DISPOSITIVOS MÓVEIS

Um sistema operativo móvel é um sistema operativo desenvolvido para funcionar em *smartphones*, *tablets*, *PDAs* ou em qualquer outro dispositivo móvel. Os sistemas operativos móveis modernos combinam as características de um sistema operativo de computador pessoal com os restantes recursos disponibilizados pelas plataformas móveis, incluindo ecrã táctil, telefonia sem fios, Bluetooth, Wi-Fi, GPS para navegação móvel, câmara fotográfica e de vídeo, reconhecimento e gravação de voz, leitor de música, comunicação por campo de proximidade (NFC) entre outros.

2.8.1. ANDROID

A Android Inc. é uma empresa fundada em Palo Alto, na Califórnia, em Outubro de 2003 para desenvolver um sistema operativo para dispositivos móveis inteligentes, que fossem mais conscientes da localização e preferências do seu proprietário. Inicialmente o sistema operativo foi desenvolvido para câmaras digitais, mas ao perceberem que o mercado para os dispositivos não era grande o suficiente, a empresa desviou seus esforços para produzir um sistema operativo para smartphones que rivalizasse com o Symbian e o Microsoft Windows Mobile [29].

A Google adquiriu a Android Inc. em 17 de Agosto de 2005 e em 5 de Novembro de 2007, revela-se a Open Handset Alliance, um consórcio de empresas de tecnologia (incluindo a Google), fabricantes de dispositivos, operadoras de telefones móveis e fabricantes de *chipset*, com o objectivo de desenvolver padrões abertos para dispositivos móveis [30].

2.8.2. BLACKBERRY

BlackBerry 10 é um sistema operativo móvel proprietário de código fechado desenvolvido pela BlackBerry Ltd. para sua linha BlackBerry de *smartphones* e *tablets*. O BlackBerry 10 disponibiliza mais de 100 bibliotecas de código aberto e é certificado POSIX. Suporta HTML5 bem como aplicações nativas via SDK. Através da biblioteca “Android Runtime” da plataforma BlackBerry é possível executar aplicações desenvolvidas para a plataforma Android, sendo que a partir da versão 10.2.1, os arquivos APK podem ser instalados directamente sem necessidade de *wrappers* [31].

2.8.3. iOS

iOS é um sistema operativo móvel desenvolvido pela Apple Inc. e distribuído exclusivamente para hardware Apple. O iOS é um sistema operativo de código fechado e proprietário e construído sobre Darwin Core OS que é código aberto. Disponibilizado originalmente em 2007, para iPhone, tendo sido de seguida alargado para abranger outros dispositivos Apple. O iOS partilha com OS X algumas bibliotecas, tais como Core Foundation e a Foundation. O seu *toolkit* UI é o Cocoa Touch em vez do Cocoa no OS X, não sendo, portanto, compatível com as aplicações OS X. O iOS também partilha a Darwin Foundation com o OS X, mas não disponibiliza uma linha de comandos Unix aos utilizadores, ficando restringido às aplicações, o que torna o iOS apenas parcialmente compatível com o Unix. No iOS, existem quatro camadas de abstracção: a camada de Core OS, a camada Core Services, a camada multimédia e a camada Cocoa Touch [32].

2.8.4. WINDOWS PHONE

O Windows Phone é um sistema operativo móvel de código fechado e proprietário desenvolvido pela Microsoft. É o sucessor do sistema operativo Windows Mobile apesar de ser incompatível com a plataforma anterior [33]. Os dispositivos Windows Phone são fabricados principalmente pela Nokia, HTC, Samsung, Huawei e outros OEMs. David Treadwell, Microsoft CVP para a área de sistemas operativos, anunciou que o ecossistema Windows, terá aplicações universais graças ao “Windows Runtime” e este estar a chegar ao Windows Phone 8.1. Os programadores serão então capazes de escrever uma aplicação usando o código comum e tê-lo funcionar em telefones, tablets, PCs, e até mesmo na Xbox One [34].

2.8.5. FIREFOX OS

O Firefox OS é um sistema operativo móvel de código aberto mantido pela organização sem fins lucrativos Mozilla Foundation. Anteriormente chamado Boot2Gecko, o sistema operativo móvel da Mozilla é construído inteiramente sobre padrões web abertos [35]. Cada aplicação será feita inteiramente em HTML5 e o sistema operativo permitirá que as aplicações baseadas em padrões web possam aceder aos arquivos do sistema, algo que apenas as aplicações nativas podem fazer noutros sistemas operativos móveis [36]. A Mozilla acredita que a “remoção de camadas de middleware

desnecessárias” e a optimização para dispositivos de entrada de gama, vai permitir aos operadores abranger uma gama baixa de preços, ajudando a impulsionar a adopção nos mercados em desenvolvimento [37].

2.8.6. SAILFISH

O Sailfish é um sistema operativo móvel de código aberto que adopta o GPL, foi criado por ex-programadores da equipa MeeGo que saíram da Nokia em 2011 quando a empresa parou com o projecto MeeGo. Os ex-programadores formaram a sua própria empresa, designada de Jolla e poucos meses depois, apresentaram o seu primeiro dispositivo Sailfish Jolla [38]. A Jolla optou por uma interface baseada em gestos em vez de botões no ecrã, de modo a que quando os utilizadores se acostumarem com os gestos possam economizar tempo. O sistema operativo é compatível com aplicações Android graças à camada Alien Dalvik da empresa Myriad Group AG [39]. O Sailfish também tem compatibilidade a nível de API com aplicações Ubuntu [40].

2.8.7. TIZEN

O Tizen é um sistema operativo baseado no *kernel* do Linux e tem na implementação da API Linux a biblioteca GNU C Library. O Tizen é um projecto no âmbito da Linux Foundation e é gerido por um Grupo de Coordenação Técnica (TSG) composto entre outros pela Samsung e a Intel. Embora a Associação Tizen decida o que precisa ser feito no Tizen, é o Director Técnico do Grupo que determina que código é realmente incorporado no sistema operativo [41]. O Tizen tem origens no Samsung Linux Platform (SLP) e no Projecto LiMo tendo sido incorporado em 2013 o projecto Bada pela Samsung [42]. O seu modelo de licenciamento envolve *software* com uma variedade de licenças que podem ser incompatíveis com a Open Source Definition. As interfaces de programação das aplicações Tizen são baseadas em HTML5 e noutros padrões web que poderão tirar partido das diversas capacidades da plataforma, tais como mensagens, multimédia, câmara, rede, etc... [43] Para as aplicações que necessitem de utilizar código nativo, o SDK do Tizen disponibiliza um kit de desenvolvimento nativo. Na primeira semana de Outubro de 2013 a câmara inteligente da Samsung NX300M tornou-se o primeiro produto para o mercado de consumo com base no Tizen [44].

2.8.8. UBUNTU TOUCH

O Ubuntu Touch é uma versão móvel do sistema operativo Ubuntu desenvolvida pela Canonical UK Ltd. e pela Comunidade Ubuntu. A Canonical disponibilizou a primeira versão do Ubuntu touch 1.0 em 17 de Outubro de 2013 em conjunto com o Ubuntu 13.10 para um grupo restrito de terminais móveis [45], e disponibilizou uma versão mais “estável” para um teste de maior amplitude e de feedback em 17 de Abril de 2014 em conjunto com o Ubuntu 14.04. O sistema operativo é construído sobre o *kernel* Linux do Android, usa os drivers do Android, mas não usa nenhum código Java do Android [46]. Os programadores têm acesso a todo o código-fonte sob uma licença que permite a modificação e redistribuição do *software*.

2.9. UTILIZAÇÃO DE INTERNET

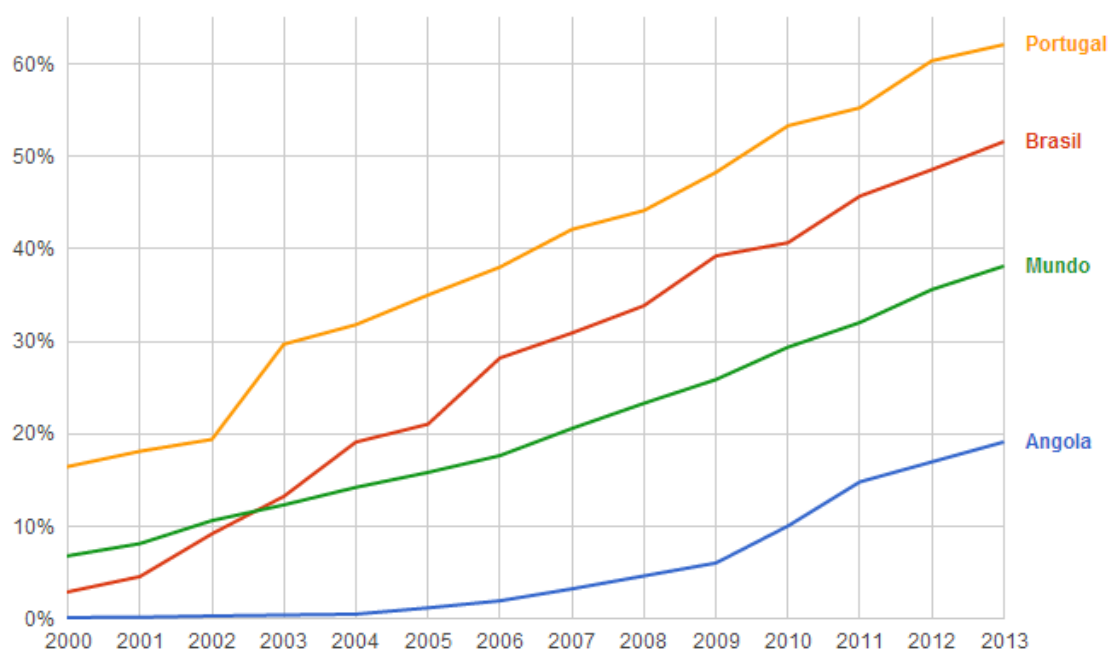


Figura 4 – Percentagem de utilizadores da Internet entre 2000 e 2013 [47]

Como se observa facilmente na Figura anterior, cada vez existem mais utilizadores da Internet em percentagem da população.

Em Portugal a penetração da Internet através do telemóvel cresceu 5 pontos percentuais no primeiro trimestre de 2014, onde cerca de 32,8 por cento dos inquiridos tinham Internet através do telemóvel. O número de utilizadores de Internet móvel com ligação através de placa/modem tem vindo a diminuir, por outro lado, a taxa de penetração de *smartphones* atingiu 41,8 por cento em Março de 2013. No trimestre de Agosto de 2014,

foram contabilizados 4134 mil indivíduos que costumam utilizar *smartphone*, o que corresponde a 46.4% dos possuidores de telemóvel residentes em Portugal com pelo menos 10 anos e a um aumento da taxa de penetração de *smartphones* de 80% relativamente ao observado em Maio de 2012 conforme pode ser observado na figura seguinte [48].



Figura 5 – Percentagem de detentores de telemóvel de Maio 2012 a Agosto 2014 [48].

2.10. EQUIPAMENTOS COMPUTACIONAIS

O ecossistema que constitui um sistema distribuído é constituído por diversos tipos de terminais móveis bem como terminais fixos, dificilmente se consegue abranger a totalidade de sistemas operativos, arquitecturas de processadores. Para a identificação e avaliação de oportunidades, uma análise dos segmentos do mercado e selecção de mercados-alvo, pode-se utilizar previsões como forma de reduzir a incerteza.

Segundo a Gartner (A empresa de consultoria líder mundial em investigação na área de tecnologias da informação), e como se pode verificar na Tabela 1, em 2015 as vendas de *tablets* vão ultrapassar as dos tradicionais computadores de mesa e computadores portáteis, embora a rápida taxa de crescimento para o sector irá abrandar. A Gartner diz que a relativa desaceleração nas vendas de *tablets* é devido a um mercado em maturação, bem como ao crescimento dos telefones celulares de ecrã grande '*phablet*', particularmente no sudeste da Ásia [49].

A Gartner defende também que a próxima onda de adoção será impulsionada por preços baixos em vez de funcionalidade superior, advertindo que o crescimento dos *tablets* será focado no segmento inferior do mercado [49].

Tabela 1 – Fabrico mundial de dispositivos por segmento [49].

Tipo de dispositivo	2013	2014	2015
PCs Tradicionais (<i>Desktop</i> ou <i>Notebook</i>)	296,131	276,221	261,657
<i>Ultramobiles, Premium</i>	21,517	32,251	55,032
PC Market Total	317,648	308,472	316,689
<i>Tablets</i>	206,807	256,308	320,964
Telemóveis	1,806,964	1,862,766	1,946,456
Outros <i>Ultramobiles</i> (Híbridos e <i>Clamshell</i>)	2,981	5,381	7,645
Total	2,334,400	2,432,927	2,591,753

Em milhares de unidades

No mercado de sistemas operativos, de acordo com o demonstrado na Tabela 2, o Android e iOS estão a impulsionar o crescimento com um aumento de 30 por cento e 15 por cento respectivamente, em 2014 [49].

Tabela 2 – Distribuição mundial de dispositivos por sistema operativo [49]

Sistema Operativo	2013	2014	2015
Android	898,944	1,168,282	1,370,893
Windows	326,060	333,419	373,694
iOS/Mac OS	236,200	271,115	301,349
Outros	873,195	660,112	545,817
Total	2,334,400	2,432,927	2,591,753

Em milhares de unidades

Os valores apresentados na Tabela 1 e Tabela 2 levam a que sejamos capazes de identificar o Android como sendo o sistema operativo com potencialmente maior número de dispositivos, e claramente se consegue identificar que o número de dispositivos móveis será bastante superior ao de dispositivos convencionais, sendo mais de metade deles dispositivos Android.

3. *FRAMEWORKS*

No capítulo anterior foram identificadas as tendências de mercado, onde foi analisado o mercado de consumo, prevendo as tecnologias que iriam cair em desuso bem como os novos *players* de mercado.

Relativamente a sistemas operativos, o Android foi identificado como sendo o sistema operativo com potencialmente maior número de dispositivos activos em 2015, sendo, portanto, o ambiente móvel a analisar.

Relativamente a protocolos, o XML-RPC e o JSON-RPC foram identificados como sendo os que cumprem com maior número de requisitos de escolha, mas dada a limitação do XML-RPC de funcionar apenas no protocolo HTTP, faz com que o protocolo a analisar mais profundamente seja o JSON-RPC.

Relativamente á camada de aplicacional, analisando o desempenho do HTTP e do MQTT, tendo em consideração as características dos terminais móveis, de onde se observa que o MQTT dispõe de maior performance, é mais leve e tem várias características que foram pensadas para serem aplicáveis a sistemas embutidos.

3.1. ANDROID

A Open Handset Alliance TM é um grupo de líderes em tecnologia móvel que defende a transparência do ecossistema móvel, e acredita que isso irá permitir que a indústria inove mais rapidamente por forma a responder melhor às exigências dos consumidores. O primeiro projecto conjunto desta aliança é o Android TM. O Android foi construído a partir do zero com o objectivo explícito de ser a primeira plataforma aberta, completa e livre criada especificamente para dispositivos móveis [50]. O licenciamento de código aberto do Android dá a liberdade para desenvolver aplicações sem custos com royalties e licenciamento. Não existe custo de adesão, de testes e taxas de certificação digital envolvidos no desenvolvimento de aplicações Android. O SDK do Android fornece as ferramentas e APIs necessárias para começar a desenvolver aplicações na plataforma Android utilizando a linguagem de programação Java [51]. Os programadores podem escolher os ambientes de desenvolvimento integrado populares para o desenvolvimento, existindo *plugins* de integração para os ambientes mais populares tais como o eclipse, netbeans, inteliJ Idea e existem dois pacotes pré configurados disponibilizados pela Android, o Android Studio baseado no inteliJ Idea e o Android developer tools baseado no eclipse.

3.2. ARQUITECTURA ANDROID

O Android é um conjunto de *software* móvel de código aberto, destinado ao mercado de consumo. Trata-se de um sistema operativo móvel completo, que tem como base o *kernel* do Linux, uma máquina virtual e algumas aplicações para terminais móveis.

A arquitectura do sistema Android consiste em aplicações, na *framework*, em bibliotecas nativas C/C++, no Runtime Android (que consiste na máquina de execução virtual e nas bibliotecas core Android que reflectem as funcionalidades das bibliotecas cores numa interface em Java), e, finalmente, o *kernel* do Linux utilizado para gerir os recursos de baixo nível [52].

Existem várias camadas no *stack* do Android, cada uma com responsabilidades diferentes, conforme o ilustrado na Figura 6, que irão ser analisadas de seguida

- Núcleo (Linux *Kernel*)
- Bibliotecas (Libraries)
- Aplicações (Applications)
- *Framework* aplicacional (Application *Framework*)

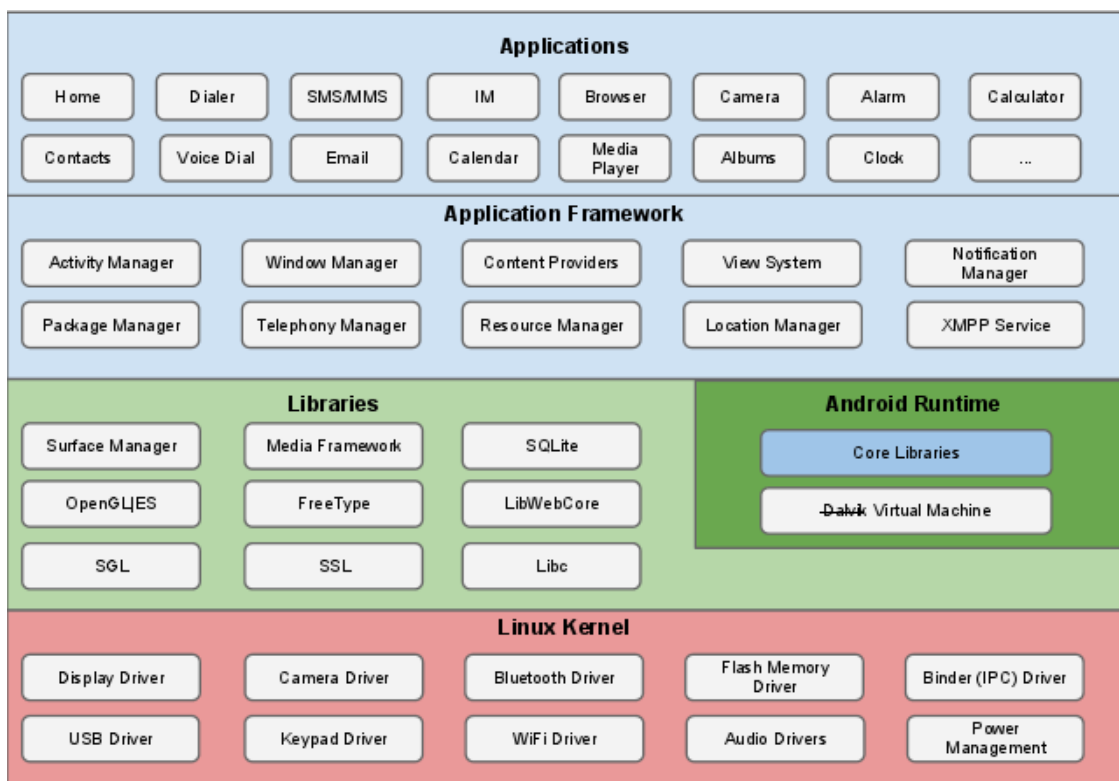


Figura 6 – Diagrama de blocos da arquitectura Android [53]

3.2.1. CAMADA DO NÚCLEO



Figura 7 – Diagrama de blocos do Núcleo do Android

Na base do modelo do Android está o seu núcleo, ilustrado na Figura 7, que é baseado no *Kernel LTS* do Linux, actualmente as versões do Android estão a ser baseadas na

versão 3.4 do *Kernel* Linux apesar de existirem fabricantes já com a versão 3.10, dependendo do suporte do dispositivo e *chipset* [54].

O Android já utilizou vários núcleos desde a sua primeira versão, seguem na Tabela 3, os nomes de código correspondentes ao número de versão e nível de API e de NDK e relacionados com a versão do núcleo do Linux [55]:

Tabela 3 – Diversas versões Android [55]

Nome de código	Versão	Nível API	Linux <i>Kernel</i>
(no code name)	1.0	API level 1	2.6.25
(no code name)	1.1	API level 2	2.6.25
Cupcake	1.5	API level 3, NDK 1	2.6.27
Donut	1.6	API level 4, NDK 2	2.6.29
Eclair	2.0	API level 5	2.6.29
Eclair	2.0.1	API level 6	2.6.29
Eclair	2.1	API level 7, NDK 3	2.6.29
Froyo	2.2.x	API level 8, NDK 4	2.6.32
Gingerbread	2.3 - 2.3.2	API level 9, NDK 5	2.6.35
Gingerbread	2.3.3 - 2.3.7	API level 10	2.6.35
Honeycomb	3.0	API level 11	2.6.36
Honeycomb	3.1	API level 12, NDK 6	2.6.36
Honeycomb	3.2.x	API level 13	2.6.36
Ice Cream Sandwich	4.0.1 - 4.0.2	API level 14, NDK 7	3.0.1
Ice Cream Sandwich	4.0.3 - 4.0.4	API level 15, NDK 8	3.0.1
Jelly Bean	4.1.x	API level 16	3.0.31
Jelly Bean	4.2.x	API level 17	3.4.0
Jelly Bean	4.3.x	API level 18	3.4.x
KitKat	4.4 - 4.4.4	API level 19	3.x.x
KitKat Wear	4.4.W	API level 20	3.x.x
Lollipop	5	API level 21	3.x.x
Lollipop MR1	5.1	API level 22	3.x.x
Marshmallow	6	API level 23	3.x.x
Nutella	7	API level 24	3.x.x

Os utilizadores e programadores não interagem directamente com o núcleo, mas não deixa de ser a parte mais importante do modelo, já que disponibiliza as seguintes funcionalidades do sistema Linux [53]:

- **Drivers e Abstracção de Hardware** -. A sua função é ocultar diferenças de *hardware* de modo a disponibilizar uma plataforma consistente que permita a descoberta e utilização do *hardware* do sistema através de uma API abstracta. O driver fornece essa interface de *software* para aceder às funções do *hardware* [52].
- **Gestão de memória** – O núcleo possui acesso completo à memória do sistema e deve permitir que processos acessem à memória com segurança conforme a sua necessidade permitindo a cada um deles que funcione como se fosse o único em execução [56].
- **Definições de segurança** – O núcleo permite ao proprietário de um objecto (por exemplo, um ficheiro) definir a sua política de segurança, onde pode, por exemplo, criar um novo ficheiro na sua directoria pessoal e decidir quem o pode ler ou gravar. Os processos lançados por um utilizador executam com todos os direitos desse utilizador, quer eles sejam precisos ou não [57].
- **Gestão de energia** – Na primeira versão do Android era utilizado um conjunto de alterações ao *kernel* do Linux designado de “*wakelocks*”. A utilização de *wakelock* permite indicar ao *kernel* a existência dum processo que quer impedir o sistema de entrar no modo de baixo consumo de energia [58]. Na versão 3.3 do núcleo é possível inicializar um *userspace* Android sem modificações, mas com fracas funcionalidades de gestão de energia, algo que já está solucionado na versão 3.4 [59].
- **Suporte para bibliotecas partilhadas** [52] – Bibliotecas partilhadas permitem aos executáveis aceder dinamicamente a funcionalidades externas em tempo de execução e a reduzir a sua taxa de ocupação de memória. Vários programas podem usar uma biblioteca partilhada simultaneamente; sendo, portanto, apenas necessária uma cópia da biblioteca em memória. Com uma biblioteca estática, cada programa em execução tem sua própria cópia da biblioteca [60].

- **Stack de rede** – Permite o acesso à rede por meio de um dispositivo de rede, disponibiliza a System Call Interface (SCI) onde é feita a chamada, no Protocol Agnostic Interface é criado e gerido o *socket*, enquanto que na camada de protocolo de rede é gerida a forma como os dados são enviados ou recebidos para a rede [61].

3.2.1.1. COMUNICAÇÃO ENTRE PROCESSOS

Ao contrário do núcleo padrão do Linux, o mecanismo de IPC no núcleo do Android, designado de binder, baseia-se no OpenBinder que é utilizada para IPC no sistema operativo BeOS por forma a ser um mecanismo mais leve que o System V [62].

As Aplicações e os Serviços podem ser executados em processos separados e mesmo assim devem conseguir comunicar e partilhar dados. A utilização de um mecanismo de comunicação entre processos, pode trazer um aumento significativa de processamento e algumas falhas de segurança. O driver *binder* utiliza memória partilhada para passar mensagens entre as *threads* e processos de modo a aumentar o desempenho e ao mesmo tempo facilitando a comunicação entre processos (IPC) [63].

Como a *framework binder* utiliza uma *pool* de *threads* por processo para o processamento de pedidos e memória partilhada obtém um elevado desempenho. A *framework binder* suporta também chamadas síncronas, contagem e mapeamento de referências de objectos entre de processos. Na Figura 8 está ilustrado sob a forma de um diagrama o funcionamento do *driver binder*

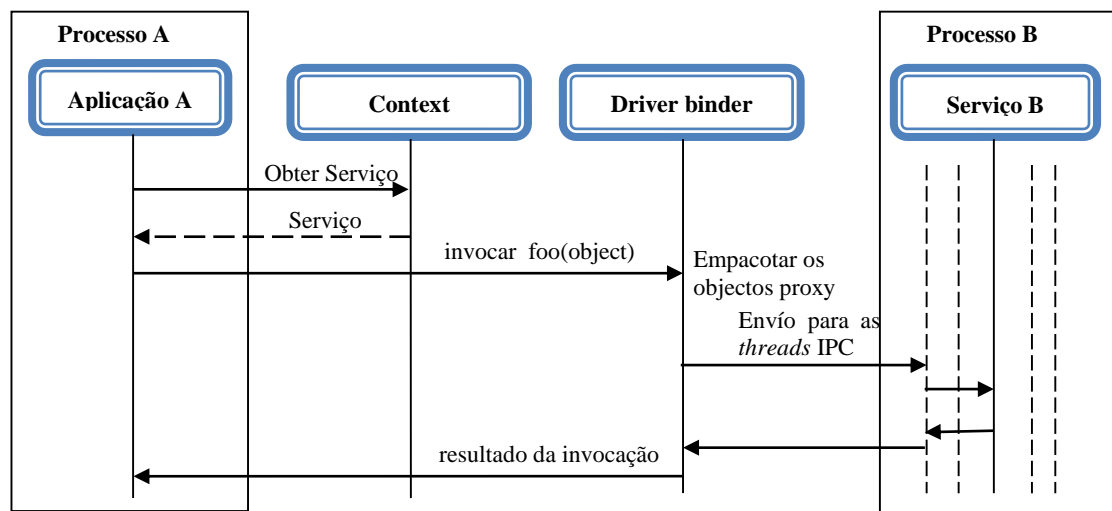


Figura 8 – Funcionamento do *driver binder* [62]

3.2.2. CAMADA DE BIBLIOTECAS



Figura 9 – Diagrama de blocos da camada de bibliotecas

A camada seguinte na arquitetura Android inclui as bibliotecas nativas Android, algumas delas são apresentadas na Figura 9. As bibliotecas contêm um conjunto de instruções que levam o dispositivo a gerir diferentes tipos de dados. Por exemplo a reprodução e gravação de vários formatos de vídeo e de áudio é gerido pela Biblioteca de *Media Framework* [64].

Para desenvolver bibliotecas de uma aplicação Android é necessário a instalação do Native Development Kit (NDK) do Android, mas não do SDK. O NDK suporta a linguagem C e C++ enquanto o SDK suporta linguagem Java [65].

Todas as bibliotecas são escritas em linguagem C/C++.

3.2.2.1. BIBLIOTECAS DE CÓDIGO ABERTO

- **Surface Manager:** [66] Fornece um compositor de superfície a todo o sistema “manuseando” toda a renderização da superfície antes de a enviar para o dispositivo *frame buffer*. Pode combinar superfícies 2D e 3D e superfícies a partir de múltiplas aplicações
- **Media Framework:** Biblioteca baseada na plataforma PacketVideo OpenCORE para dar suporte à reprodução e gravação de vários formatos de áudio, vídeo e imagem [64].
- **SQLite:** Um leve motor de base de dados transaccional, é o responsável pela maioria do armazenamento de dados da plataforma [67]
- **Open GL|ES:** Biblioteca múltipla plataforma que suporta gráficos 2D e 3D em sistemas embutidos, que consiste num subconjunto bem definidos do OpenGL [68].

- **Free Type:** Renderização de Fontes [69]
- **WebKit:** Motor do navegador de internet, renderiza páginas em ecrã completo e tem suporte de CSS, Javascript, DOM e AJAX [70]
- **SGL:** (Scalable/Skia Graphics Library) é uma biblioteca que lida com a renderização de Texto, Geometrias e Imagens. SGL é compatível com os padrões 2D existentes [71]
- **SSL:** biblioteca baseada no OpenSSL, disponibiliza uma biblioteca de criptografia e uma implementação dos protocolos *Secure Sockets Layer* (SSL v2/v3) e *Transport Layer Security* (TLS v1) [72].
- **libc:** Biblioteca bionic, consiste numa implementação personalizada da biblioteca libc otimizada para funcionar em dispositivos embutidos [73].

No mesmo nível da arquitectura encontra-se o Runtime Android que inclui um conjunto de bibliotecas Java, que os programadores Android utilizam para construir as suas aplicações na linguagem de programação Java. Neste nível também está incluída a Máquina Virtual.

3.2.3. RUNTIME ANDROID

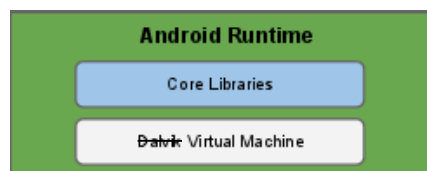


Figura 10 – Diagrama de blocos do Runtime Android

O bloco seguinte da arquitectura Android é o Android Runtime que está ilustrado na Figura 10, que é o responsável pela execução do *software*. Quando se está a executar o *software* num computador, podemos correr dois tipos de código, o código nativo ou o código interpretado. O código nativo é o código fonte que foi compilado, por ferramentas específicas de uma arquitectura de processador e plataforma, transformado num código que o processador possa executar (código de máquina). O outro tipo de código é o código interpretado que não pode ser executado directamente pelo processador, deve ser compilado pelo interpretador em código de máquina. O problema

do código interpretado é que é bastante mais lento que o código nativo. A vantagem do código interpretado é que um programador pode escrever um programa uma vez, e depois tê-lo executado em qualquer plataforma de hardware que tenha um interpretador de código [74]. Como o Android foi projectado para correr numa grande variedade de plataformas, especificações de *hardware*, e factores de forma, o Google decidiu usar a máquina virtual Dalvik para aplicações Android. Desta forma, um programador pode escrever uma única aplicação Dalvik que ela será executada em smartphones, tablets, TVs, dispositivos embutidos, etc. [75].

3.2.4. DALVIK VIRTUAL MACHINE

Dalvik é mais um bloco de *software* de código aberto criado por Dan Bornstein, que lhe deu o nome da aldeia piscatória de Dalvík em Eyjafjörður, Islândia de onde eram originários alguns dos seus antepassados. A máquina virtual Dalvik é responsável pelo ambiente de execução das aplicações nos dispositivos Android.

O Dalvik é uma máquina virtual baseada em registos, e optimizada para ter necessidade de pouca memória, que foi desenhada para permitir que várias instâncias sejam executadas em simultâneo. Utiliza o sistema operativo subjacente para o isolamento de processos, gestão de memória e suporte de threads. [76]

O Dalvik executa sobre ficheiros do tipo Dalvik Executable (DEX).

3.2.5. ANDROID RUNTIME (ART)

Com Android 4.4, a Google lançou o Android RunTime (ART) como sendo um ambiente de execução opcional no Android 4.4 e passou a ser o padrão para as versões seguintes [77].

Durante o processo de instalação da aplicação num dispositivo Android, o ART aplica a técnica de compilação Ahead Of Time, onde o código interpretado é traduzido para código nativo. A ferramenta dex2oat analisa o arquivo DEX e recompila-o para o Extensible Linking Format (ELF). Esse arquivo consiste no código DEX no resultado da sua recompilação, no código nativo existente e nos meta-dados. O dex2oat ao manter o código de DEX original vai permitir que ferramentas existentes ainda funcionem. Este processo resulta num aumento de aproximadamente 30% no tamanho dos ficheiros, mas

permite uma execução mais rápida a partir do início da aplicação. É uma equação simples de fazer: o código interpretado leva mais tempo para executar logo consome mais tempo de CPU, pelo que vai reduzir a vida útil da bateria bem como a capacidade global de resposta [78].

O *garbage collector* no ART também foi otimizado para reduzir os tempos em que o pedido pára. No Dalvik, este processo necessitava de duas pausas e 10 milissegundos para completar a colecta de lixo, no ART é apenas necessária uma pausa e o processo de colecta de lixo termina em menos de 2 milissegundos. O *garbage collector* no ART tem em conta as características da plataforma Android, como por exemplo o ciclo de vida das aplicações e das *activities*, efectuando portanto a colecta completa apenas quando o telefone está bloqueado e noutras situações onde a capacidade de resposta à interacção do utilizador não é importante [79].

O ART utiliza um novo alocador de memória designado de *rosalloc* em oposição ao Dalvik que tem uma única região de memória global. Com o *rosalloc*, os objectos menores são alocados numa região *thread local* sem bloqueios e os objectos maiores têm seus próprios bloqueios. Assim, quando a aplicação tenta alocar memória para novos objectos não tem que esperar que o colector de lixo liberte regiões não relacionadas de memória [79].

3.2.6. CAMADA DA *FRAMEWORK* APLICACIONAL



Figura 11 – Diagrama de blocos da *Framework* Aplicacional

As aplicações interagem directamente com estes blocos da arquitectura Android. Os blocos de *software* gerem funcionalidades básicas, tais como gestão de recursos e gestão de chamadas, etc.

Os blocos importantes da *framework* aplicacional, que estão representados na Figura 11, têm as seguintes funcionalidades:

- **Activity Manager:** Gere o ciclo de vida das aplicações e das Activities [80].

- **Content Providers:** Gere a partilha de dados entre aplicações. Encapsula os dados e fornece mecanismos para definir a segurança de dados. Os *content providers* disponibilizam a interface padrão para aceder a dados num processo com o código em execução noutro processo [81].
- **Telephony Manager:** Gere as chamadas de voz, por exemplo se for necessário efectuar chamadas de voz de dentro da aplicação teremos de o utilizar [82].
- **Location Manager:** Gestão de localização, utilizando as antenas de telemóveis ou GPS [83]
- **Resource Manager:** Gere os diversos tipos de recurso que utiliza na aplicação.

3.2.7. CAMADA APLICACIONAL

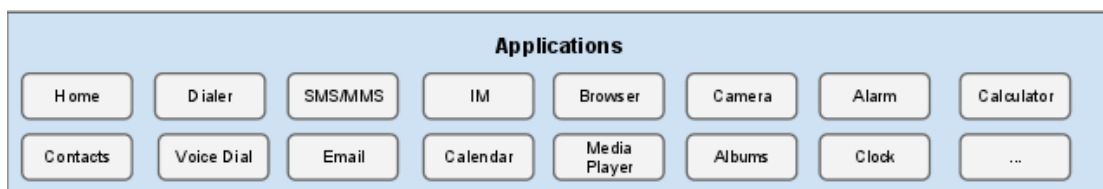


Figura 12 – Diagrama de blocos da camada das Aplicações

As aplicações encontram-se no topo do modelo. Um utilizador típico do dispositivo Android interage a maioria das vezes com esta camada (para funcionalidades básicas tais como efectuar chamadas ou aceder ao navegador de internet, etc.).

Nesta camada são disponibilizadas aplicações normalmente utilizadas no dia-a-dia de um dispositivo móvel, tais como: Cliente de Mensagens escritas (SMS), Marcador telefónico, Navegador de internet, Gestor de contactos. A figura 12 mostra algumas destas aplicações.

3.3. DESENVOLVIMENTO ANDROID

O Android Software Development Kit (Android SDK) contém as ferramentas necessárias para criar, compilar e empacotar as aplicações para Android. A maioria destas ferramentas é baseada em linha de comando. Uma dessas ferramentas é o Android Debug Bridge (ADB), que é uma ferramenta que permite a ligação a um dispositivo Android (real ou virtual), com a finalidade de o gerir ou depurar a sua aplicação [84].

3.3.1. AMBIENTES DE DESENVOLVIMENTO INTEGRADO

Os ambientes integrados de desenvolvimento (IDEs) devem disponibilizar as funcionalidades necessárias para criar, compilar, depurar e implantar aplicações Android, bem como permitir que o programador possa criar e iniciar os dispositivos virtuais Android para testes, bem como edição especializada nos ficheiros específicos Android. A maioria dos ficheiros de configuração do Android são baseados em XML os editores permitem que se alterne entre a representação XML e a interface de utilizador para inserir os dados. A Google disponibiliza dois IDE para facilitar o desenvolvimento de novas aplicações. O Android Developer Tools (ADT) [85] que estende o IDE Eclipse com um conjunto de componentes e o IDE Android Studio [86] que é baseado no IDE IntelliJ IDEA.

3.3.2. CICLO DE DESENVOLVIMENTO DE APLICAÇÕES PARA ANDROID

Durante o ciclo de desenvolvimento são criados vários arquivos de configuração específicos para o Android e é escrita a lógica da aplicação na linguagem de programação Java. A lógica da aplicação é convertida em arquivos binários (.class) pelo compilador Java. É utilizada uma ferramenta do Android SDK designada de dx que vai converter todos os arquivos .class da aplicação em arquivos DEX (Dalvik Executable) [87]. Durante este processo de conversão, todas as informações redundantes nos arquivos de .class são optimizadas no arquivo DEX. Por exemplo, se a mesma sequência de caracteres é encontrado em diferentes arquivos de classe, o arquivo DEX contém apenas uma referência desta cadeia [76]. Esses arquivos DEX são muito menores em tamanho do que os arquivos .class correspondente. O arquivo DEX e os recursos (por exemplo, as imagens e os arquivos XML), são embalados dentro de um arquivo APK (pacote Android). O SDK Android de seguida utiliza uma ferramenta chamada AAPT (Android Asset Packaging Tool) para realizar esta etapa. O arquivo APK resultante contém todos os dados necessários para executar a aplicação Android e poder ser instalado num dispositivo pela ferramenta ADB. A Figura 13 ilustra esse processo

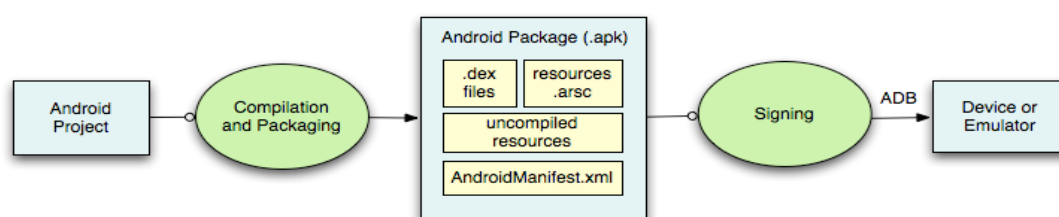


Figura 13 – Processo de instalação de uma aplicação Android [87]

3.4. JSON – JAVASCRIPT OBJECT NOTATION

JSON é um formato aberto de troca de dados baseado em texto, criado por Douglas Crockford, e encontra-se descrito no RFC4627 [88], pelo que pode ser utilizado em praticamente qualquer cenário em que as aplicações precisam trocar ou armazenar informações estruturadas em texto. Tal como o XML, é legível, independente de plataforma e oferece diversas implementações em diversas plataformas e linguagens de programação. O JSON é um formato de troca de dados que foi criado a partir de um subconjunto da notação literal de objecto em JavaScript, pelo que os dados formatados de acordo com o padrão JSON podem ser analisados facilmente pelas diversas implementações de JavaScript.

O JSON pode representar quatro tipos primitivos (*strings*, números, booleanos e nulos) e dois tipos estruturados (objectos e *arrays*). Uma *string* é uma sequência de zero ou mais caracteres *Unicode*, onde a codificação padrão é UTF-8. Um objecto é uma colecção não-ordenada de zero ou mais pares nome/valor, onde um nome é uma *string* e o valor pode ser uma *string*, um número, um booleano, *null*, um objecto ou um *array*. Um *array* é uma sequência ordenada de zero ou mais valores ou objectos [88].

O JSON tem regras muito mais rigorosas do que a sintaxe aceite pelo JavaScript, por exemplo, o nome de um membro do objecto deve ser uma *string*. Uma *string* em JSON é uma sequência de caracteres que deve ser colocada entre aspas. O JavaScript permite que os nomes de membros do objecto sejam delimitados por aspas ou apóstrofes ou não delimitados (desde que o nome do membro não entre em conflito com uma palavra-chave reservada do JavaScript) [88].

Uma mensagem no formato JSON é composta por um único objecto ou um vector. Os elementos da vector e valores do objecto podem ser objectos, *arrays*, *strings*, números ou nulo.

3.5. JSON-RPC

JSON-RPC é a especificação de um mecanismo leve para utilização em sistemas informáticos, para a execução remota de código. Este mecanismo é agnóstico ao transporte, sendo que os objectos JSON transportam toda a informação necessária para que a aplicação destino saiba quais os métodos a correr com que parâmetros. O cliente neste processo é tipicamente o *software* que tem a intenção de chamar o método do sistema remoto. Podem ser passados vários parâmetros de entrada para o método remoto, na forma de um vector ou de um objecto, e o método pode retornar vários dados de saída também nesse formato [89].

Todos os tipos de dados de transferência são objectos únicos, serializados usando JSON. A solicitação é uma chamada a um método específico fornecido por um sistema remoto. Ele deve conter três propriedades:

method uma *string* com o nome do método a ser invocado.

params Um vector de objectos a serem passados como parâmetros para o método definido.

id Um valor de qualquer tipo, que é usado para relacionar a resposta com o pedido ao qual está a responder.

O receptor do pedido deve responder com uma resposta válida para os pedidos recebidos. A resposta deve conter as seguintes propriedades:

result Os dados retornados pelo método invocado. Se ocorreu um erro ao chamar o método, este valor deve ser nulo.

error Um código de erro especificado se houve um erro de chamar o método, caso contrário este valor deve ser nulo.

id O id indicado no pedido de resposta.

Existem situações em que não é necessário ou desejado obter uma resposta, para estas situações foram introduzidas as notificações. A notificação é semelhante a um pedido, excepto que o id deixa de ser necessário já que nenhuma resposta será devolvida.

3.6. MQTT

O Message Queue Telemetry Transport (MQTT) é um protocolo leve de mensagens baseado num mecanismo de publicação-assinatura, desenvolvido e projectado para dispositivos restritos e baixa largura de banda pela IBM/Eurotech em 1999. Devido à simplicidade e baixo custo operacional, este protocolo é adequado para uso em ambientes constrangidos, tais como [25]:

- Redes com baixa largura de banda e sem confiabilidade.
- Incorporado em dispositivos com processamento e/ou recursos de memória limitados.

Algumas outras características deste protocolo são:

- Fornece distribuição e dissociação de mensagens de um-para-muitos.
- É agnóstico sobre o conteúdo da carga útil.
- É construído sobre TCP / IP para conectividade de rede básica.
- Oferece três tipos diferentes de Qualidade do Serviço:
 - **No máximo uma vez:** as mensagens são entregues de acordo com o melhor esforço de redes TCP / IP; pode ocorrer perda e duplicação de mensagens.
 - **Pelo menos uma vez:** as mensagens vão de certeza chegar, mas podem ocorrer duplicações.
 - **Exactamente uma vez:** as mensagens são entregues sempre só uma vez.
- Tem uma pequena sobrecarga de transporte.
- Tem um mecanismo de notificação de terminação anormal de um cliente usando funcionalidades de última vontade e de testamento.

Uma mensagem MQTT consiste em três blocos [25], conforme o ilustrado na figura 14:



Figura 14 – Formato da mensagem MQTT

O cabeçalho fixo, ilustrado na figura 15, contém informações sobre o tipo de mensagem, o nível de QoS, alguns argumentos e o tamanho da mensagem [25].

Bit	7	6	5	4	3	2	1	0
Byte 1	Message type				DUP <i>flag</i>	QoS level		RETAIN
Byte 2	Remaining Length							

Figura 15 – Formato cabeçalho fixo da mensagem MQTT

Message type: Este campo é usado para definir o tipo de mensagem. É um campo de 4 bits.

DUP flag: Este campo é definido quando o cliente ou servidor tenta retransmitir uma mensagem PUBLICAR, PUBREL, SUBSCRIBE ou UNSUBSCRIBE. O argumento é usado apenas em mensagens que têm $QoS > 0$.

QoS level: Este campo é usado para definir o nível de garantia de entrega para as mensagens de publicação.

RETAIN: Este campo é usado apenas em mensagens de publicação. Se este argumento for definido, o *broker* deve manter a mensagem depois de ter sido entregue aos assinantes actuais. Quando uma nova subscrição é estabelecida num tópico, as últimas mensagens com o campo RETAIN retidas devem ser enviadas para o assinante. Se não existirem mensagens retidas, nada acontecerá.

Remaining Length: Este campo representa os bytes restantes na mensagem actual, incluindo o cabeçalho variável e a carga útil.

Algumas mensagens de comando MQTT contêm um cabeçalho variável que reside entre o cabeçalho fixo e a carga útil. O tamanho deste cabeçalho não é contemplado dentro do campo de Remaining Length.

3.6.1. QoS – QUALITY OF SERVICE

O MQTT permite diferentes níveis de entrega de mensagens de acordo com a qualidade do serviço correspondente.

O primeiro nível de QoS é designado de “no máximo uma vez”: a mensagem é entregue com o melhor esforço do nível da rede TCP / IP subjacente. No protocolo, não há o conceito de retransmissão, pelo que a mensagem pode ou não chegar ao *broker* [25].

O segundo nível de QoS é chamado de “pelo menos uma vez”: relativamente ao QoS anterior, cada mensagem é identificada univocamente por um ID e cada mensagem recebida correctamente é confirmada por um *acknowledge*, usando uma primitiva PUBACK. Se, durante a comunicação é identificada uma falha ou o ACK não é recebido após um período de tempo especificado, o cliente envia a mensagem novamente, mas com o campo DUP activo no cabeçalho fixo. Mensagens de SUBSCRIBE e UNSUBSCRIBE são enviadas com um nível de QoS Um [25].

O terceiro nível de QoS garante que mensagens duplicadas não são entregues à aplicação de recepção. Isto aumenta o nível de Tráfego de rede, mas que poderá ser aceitável quando as aplicações requerem a recepção correcta da mensagem devido à importância do conteúdo da mesma. Como no nível anterior, as mensagens são identificadas univocamente por um ID [25].

4. *A FRAMEWORK*

Para que seja Simples Lidar com Objectos Remotos, foi desenvolvida a *framework* *Simple Remote Object Dealing* - SROD. Através da *framework* SROD deverá ser possível efectuar chamadas a objectos noutra espaço de memória sabendo apenas que o objecto receptor da chamada suporta uma determinada interface. A *framework* SROD fornece os mecanismos para que a troca de informação entre cliente e servidor seja possível, sem que seja necessário dar a conhecer o protocolo de rede. Caso seja necessário ao programador alterar o comportamento da *framework*, poderá fazê-lo.

Nesta primeira versão da *framework* vai-se deixar visível para o programador um pouco das opções de rede e empacotamento. Numa aplicação essas opções deveriam ser definidas pelo arquitecto da solução, já que as mesmas têm um impacto directo na arquitectura da solução.

A versão aqui implementada da *framework* tem como base a linguagem de programação Java, mas devido ao tipo de protocolos utilizados, pode ser implementada noutras linguagens de programação.

A *framework* não especifica protocolo de balanceamento de carga, nem redundância do mediador, pois recorre ao protocolo MQTT que implementa essas funcionalidades, sendo uma questão de configuração do mediador a sua disponibilização e utilização.

4.1. ARQUITECTURA

Conforme o representado pela Figura 16, um servidor SROD instancia objectos remotos, torna as referências para esses objectos acessíveis e espera por clientes que invoquem os métodos desses objectos. Um cliente SROD obtém uma referência remota para um ou mais objectos remotos de um servidor e invoca um ou mais métodos. Tanto o cliente como o servidor podem estar numa rede privada ou numa rede pública, desde que consigam aceder ao mediador (*broker*) SROD.

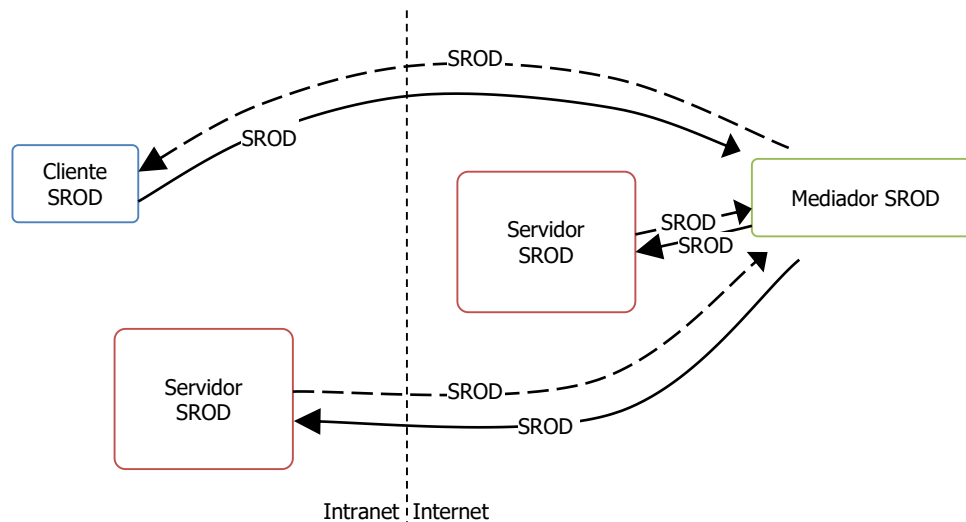


Figura 16 – Arquitectura SROD

Para a aplicação obter referências de objectos remotos deve consultar o mediador SROD, sendo este o responsável por indicar as referências dos objectos remotos. Os detalhes de comunicação entre os objectos remotos são tratados pelo SROD, ou seja, para o programador, a comunicação remota é semelhante a uma chamada a um método local.

Numa solução 100% Java, o servidor e cliente SROD irão partilhar uma interface. O Servidor irá implementar o comportamento dessa interface e pedir à *framework* que disponibilize um objecto remoto. O cliente por sua vez irá pedir à *framework* que lhe seja entregue uma implementação da interface que irá comunicar com esse objecto.

O excerto de Código 1, demonstra um caso particular de um interface partilhada de um serviço de cálculo:

```

public interface Calculable extends Serializable {
    public double add(double x, double y);
    public double multiply(double x, double y);
}

```

Código 1 – Exemplo de uma interface de serviço designada de Calculadora

Uma implementação possível do lado do servidor seria a apresentada no bloco de Código 2:

```

public class Calculator implements Calculable {
    public double add(double x, double y) {
        return x + y;
    }

    public double multiply(double x, double y) {
        return x * y;
    }
}

```

Código 2 – Exemplo da implementação da interface Calculadora

O servidor irá pedir à *framework* SROD que disponibilize a implementação, o cliente irá pedir à *framework* SROD que lhe entregue uma implementação da interface.

4.1.1. SROD - SIMPLE REMOTE OBJECT DEALING

Para o desenvolvimento da *framework* SROD optou-se pelo desenho do sistema num formato de camadas, demonstrado pela Figura 17, onde cada uma delas tem funções distintas. O desenho em camadas, permite uma maior reutilização pois possibilita a existência de implementações diferentes para uma determinada camada, permite também um desenvolvimento incremental, já que o desenho contém vários níveis de abstracção e permite uma maior evolução pois a alteração de uma determinada camada apenas vai afectar as camadas imediatamente acima e abaixo. A *framework* SROD foi projectada para funcionar com um conjunto pré-definido de protocolos, mas esta abordagem à sua arquitectura permite que outros protocolos sejam adicionados sem grande esforço.

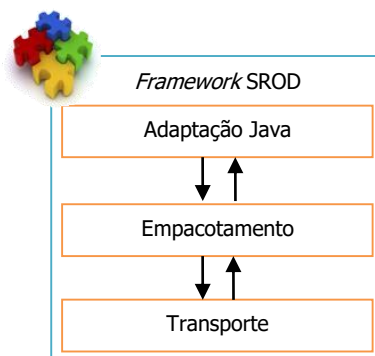


Figura 17 – Diagrama de Blocos da *Framework SROD*

De seguida virá uma explicação do funcionamento de cada uma delas, bem como os detalhes do protocolo.

4.1.1.1. CAMADA DE ADAPTAÇÃO JAVA

A camada de adaptação Java é a camada responsável por esconder os detalhes do protocolo. A camada de adaptação Java deve disponibilizar ao cliente uma implementação da interface pedida e ao servidor deve disponibilizar uma forma de publicar o objecto remoto.

Utilizando os mecanismos de reflexão da linguagem de programação Java é possível criar implementações dinâmicas das interfaces em tempo de execução para atribuir aos clientes do protocolo utilizando para tal a classe `Java.lang.reflect.Proxy`.

Utilizando o método `Proxy.newProxyInstance`, são criados os *proxies* dinâmicos. Para conseguir este comportamento são necessários 3 parâmetros:

1. O `ClassLoader` que é o carregador da classe do *proxy* dinâmico.
2. Um conjunto de interfaces a implementar.
3. Um `InvocationHandler` para redireccionar os métodos invocados no *proxy*.

Um exemplo da criação dinâmica de um proxy é apresentado na linha de Código 3:

```
TheInterface proxy = (TheInterface) Proxy.newProxyInstance(
    TheInterface.class.getClassLoader(),
    new Class[] { TheInterface.class },
    new TheInvocationHandler());
```

Código 3 – Exemplo da criação do proxy dinâmico

Após a execução destas linhas de código, o `proxy` torna-se uma referência para uma implementação dinâmica da interface `TheInterface`, onde todas as chamadas aos métodos no `proxy` dinâmico são reencaminhadas para o `handler` que é a sua implementação específica do `InvocationHandler` conforme o demonstrado no bloco de Código 4.

```
public class TheInvocationHandler implements InvocationHandler {  
    public Object invoke(Object proxy, Method method,  
                        Object[] args) throws Throwable {  
        // nop. nop. nop. I want to do something dynamic  
    }  
}
```

Código 4 – Exemplo da implementação do `InvocationHandler`

Assim, quando um método no *proxy* dinâmico é chamado, o Java vai empacotar os parâmetros e preparar uma chamada ao método `invoke` da implementação `InvocationHandler`, passando-lhe 3 parâmetros:

1. O parâmetro `proxy` é o objecto de *proxy* dinâmico que implementa a interface.
2. O parâmetro `method` representa o método do *proxy* dinâmico que foi invocado. A partir deste objecto pode-se obter o nome do método, tipos de parâmetro, tipo de retorno, etc.
3. O parâmetro `args` é um vector de objectos que contém os valores dos parâmetros passados ao *proxy* aquando a invocação do método.

O resultado da chamada ao método `invoke` será depois devolvido ao chamador como sendo o resultado da invocação.

O `InvocationHandler` irá, portanto, invocar a camada de empacotamento, para dar seguimento ao protocolo, com os seguintes parâmetros:

- id** Um valor de qualquer tipo, que é usado para relacionar a resposta com o pedido ao qual está a responder
- method** Uma *string* com o nome do método a ser invocado.

params Um vector de objectos a serem passados como parâmetros para o método a ser invocado.

endpoint Uma *string* com a localização do objecto a ser invocado.

Do lado do servidor, a camada de adaptação Java recebe um pedido da camada de empacotamento a invocar um método de um objecto com um conjunto de parâmetros.

Utilizando os mecanismos de reflexão da linguagem de programação Java, é possível invocar métodos de uma determinada classe, utilizando para tal a classe `Java.lang.reflect.Method`.

Tendo um objecto do tipo `Method` pode-se invocar o seu método `invoke()` com a instância do objecto em que este método particular é para ser invocado, e os parâmetros que esse método necessita para a sua correcta execução. Se durante a invocação do método ocorrer uma excepção, esta será envolvida por uma `Java.lang.reflect.InvocationTargetException`. A excepção original do método pode ser obtida utilizando o método do mecanismo de encadeamento de excepções `InvocationTargetException.getCause()`

O **endpoint** está relacionado com a localização de uma determinada interface, pelo que se sabe qual a classe (bem como a sua hierarquia) de terminada implementação. Utilizando o método `getDeclaredMethods()` da classe `Class`, tem-se uma forma de retornar todos os métodos explicitamente declarados nessa classe. Os métodos obtidos encontram-se devidamente caracterizados, indicando a quantidade e o tipo de dados dos seus parâmetros de entrada e o tipo de dados de retorno. Para cada classe também se pode obter a sua classe pai utilizando para tal o método `getSuperclass()`.

Tendo-se o nome do método e os parâmetros de invocação, o algoritmo irá percorrer a estrutura hierática de classes para obter o método que irá ser invocado. Poderá ter de se converter alguns parâmetros devido a algumas limitações, por parte da camada de empacotamento, que serão explicadas de seguida.

4.1.1.2. CAMADA DE EMPACOTAMENTO

A camada de empacotamento (*Marshalling*) é responsável por transformar a representação de memória de um objecto num formato de dados adequado para a sua transmissão pela camada de transporte. Esta camada também é responsável pelo processo oposto que é chamado de desempacotamento (*unmarshalling*).

O Java tem uma estrutura de tipos bastante mais rica do que o JSON, logo qualquer conversão de Java para JSON terá de incluir necessariamente algumas limitações. De modo a que se consiga converter o mais fielmente possível os objectos Java é aplicada a seguinte sequência de regras:

1. Caso o objecto Java seja nulo, é gerada a palavra **null**.
2. Caso o objecto Java seja um booleano, é gerada a palavra **true** ou **false** conforme o valor
3. Caso o objecto Java seja um número (tal como um Integer, Double, ...), é gerado um número em formato de texto.
4. Caso o objecto Java seja um caractere ou uma cadeia de caracteres, é gerada uma cadeia de caracteres.
5. Caso o objecto Java seja um Map, é gerado um objecto cujos elementos consistem nas entradas do mapa.
6. Caso o objecto Java seja um vector, ou algo iterável é gerada um vector.
7. Caso contrário, trata-se o objecto de acordo com as regras de nomenclatura dos JavaBean e é gerado um objecto cujos seus elementos consistem nas propriedades do Bean. Neste caso em particular, é necessário analisar se o nome da classe do JavaBean deve ser tratado como uma propriedade e gerar-se um atributo "class". Caso seja necessário gerar o atributo "class", pode-se utilizar o método getClass() que é fornecido por todos os objectos Java para representar a sua classe de uma forma legível.

Em Java os objectos podem referir-se a outros de uma forma cíclica. O JSON não tem suporte para referências cíclicas pelo que não é capaz de escrevê-las nativamente, sendo que se for necessário preservar essas relações, então provavelmente JSON não é um bom formato de dados a utilizar.

Analizando o excerto de Código 5 que demonstra uma referência cíclica:

```
package teste;

import java.util.ArrayList;
import java.util.List;

public class Example {
    public static class Voo {
        private String numero;
        private List<Passageiro> passageiros;
        // getters and setters omitted
    }

    public static class Passageiro {
        private String nome;
        private Voo voo;
        // getters and setters omitted
    }

    public static void main(String[] args) {
        final Voo voo = new Voo();
        voo.numero = "TP1234";
        final Passageiro passageiro = new Passageiro();
        passageiro.nome = "Sergio Rodrigues";
        passageiro.voo = voo;
        voo.passageiros = new ArrayList<>(1);
        voo.passageiros.add(passageiro);
    }
}
```

Código 5 – Exemplo de uma referência cíclica.

Este exemplo simples demonstra uma dependência cíclica, pois o voo refere-se a um passageiro, que por sua vez referencia o mesmo o voo. As dependências cíclicas como estas são bastante comuns em Java. Se for aplicada uma aproximação simples na serialização para JSON, iria-se obter algo similar a:

```

{
  "voo": {
    "numero": "TP1234",
    "passageiros": [{
      "nome": "Sergio Rodrigues",
      "voo": {
        "numero": "TP1234",
        "passageiros": [{
          "nome": "Sergio Rodrigues",
          "voo": {
            "numero": "TP1234",
            "passageiros": [{
              "nome": "Sergio Rodrigues",
              ...
            }
          ]
        }
      ]
    }
  ]
}

```

Figura 18 – Sérição JSON sem ter em consideração referências cíclicas

O ciclo nunca iria concluir, pois não há nenhuma condição de paragem. Uma forma de contornar este tipo de situação é utilizando referências aos objectos que já foram seriados, tais como:

```

{
  "voo": {
    "numero": " TP1234",
    "passageiros": [{
      "nome": "Sergio Rodrigues",
      "voo": { "href": "TP1234" }
    },
    ...
  ]
}

```

Figura 19 – Sérição JSON considerando a possibilidade de referências cíclicas

Isto contorna a questão das referências cíclicas impedindo a recursividade, mas pode introduzir outros problemas.

A *framework* SROD tenta introduzir suporte para referências cíclicas utilizando a abordagem de referências a objectos já empacotados utilizando a *keyword* “this” que é uma palavra reservada em Java, pelo que nenhum objecto pode ter um atributo com o nome “this”.

No processo de desempacotamento, é necessário converter de JSON para Java, ou seja, será necessário retornar um objecto Java resultante da análise do JSON. Tudo o que varia é o tipo do objecto retornado.

1. Caso o JSON seja um vector, é convertido para uma colecção Java, neste caso um ArrayList.
2. Caso o JSON seja um objecto, é convertido para um objecto Java do tipo Mapa, neste caso um HashMap.
3. Caso o JSON seja um numérico, é convertido para um objecto Java do tipo Number, que pode ser um dos seguintes tipos: Long, BigInteger, Double ou BigDecimal.
4. Caso o JSON seja uma cadeia de caracteres, será convertido para um objecto Java do tipo String.
5. Caso o JSON seja respectivamente o valor **true** ou **false** é convertido para um objecto Java do tipo booleano.
6. Caso o JSON seja **null**, será convertido para um objecto nulo em Java.

As regras descritas também são respeitadas por cada elemento dentro de um objecto ou de um vector JSON.

Como se verifica, a aplicação de várias regras tanto no empacotamento como no desempacotamento, pode levar a bastantes perdas de informação. Alguma dessa informação pode ser recuperada pela camada de Camada de Adaptação Java, já que os dados que serão passados à camada de transporte referem o método a ser invocado e os parâmetros de invocação.

Relativamente à camada de transporte, a camada de empacotamento invoca e é invocada com o *payload* da camada de transporte e o *endpoint* das comunicações.

4.1.1.3. CAMADA DE TRANSPORTE

A camada de transporte é responsável por transmitir os dados da camada de empacotamento.

A nível de camada de transporte, tanto o cliente como o servidor SROD comportam-se como clientes do mediador (*broker*) MQTT. É necessário que ambos consigam estabelecer uma ligação cliente para o mediador MQTT. O mediador MQTT vai ser a entidade responsável por gerir a recepção de mensagens publicadas e o respectivo envio para todos os clientes. Cada um dos clientes MQTT precisa de ter um identificador que é único entre todos os clientes que estão ligados ao *mediador* MQTT. O cliente após se ligar irá enviar uma mensagem MQTT com os dados da camada de empacotamento no *payload* para ser publicada e é aqui que é necessário a introdução de tópicos.

Para evitar o problema óbvio de cada cliente receber cada uma das mensagens publicadas por todos os outros clientes, as mensagens MQTT são publicados no que são chamados tópicos. Um tópico é uma cadeia estruturada que define um local num espaço, usando “/” como delimitador dos níveis de hierarquia dos espaços de nome (*namespace*). Um tópico pode ser, por exemplo, “/estacionamento/lugares/1/estado” ou “/carro/rodas/frente/direita” cabendo ao analista definir uma estrutura de tópicos apropriada para a tarefa a ser implementada. Os Clientes publicam num tópico absoluto, sem ambiguidade, mas podem inscrever-se nos tópicos usando *wildcards* para agregar mensagens. Um dos *wildcards* que podem utilizados é o “+” que representa um nível da hierarquia implícita, enquanto o *wildcard* “#” representa toda a árvore da hierarquia a partir desse ponto.

Tendo em conta o exemplo anterior, de um parque de estacionamento, um cliente que deseje receber publicações relativas ao estado do lugar de estacionamento 1, pode inscrever-se no tópico “estacionamento/lugares/1/estado”, mas se desejar receber publicações relativas ao estado de todos os lugares de estacionamento pode subscrever “estacionamento/lugares/+/estado”, mas se quiser receber publicações relativas a todos os lugares de estacionamento pode subscrever “estacionamento/lugares/#”.

O SROD implementa a seguinte estrutura de tópicos:

```
/<srod>/<object>/<_id>/<method>
```

Onde os vários parâmetros que constituem o *namespace* são os seguintes:

- srod** Uma *string* que é usada para distinguir os tópicos relativos a srod dos restantes, que possam estar a ser disponibilizados pelo *mediador* MQTT, esta *string* pode não ser utilizada se tal distinção não for necessária.
- object** Uma *string* com o nome do objecto a ser invocado.
- _id** Um valor de qualquer tipo, que é usado para relacionar diversas instâncias que possam estar em execução.
- method** Uma *string* com o nome do método a ser invocado.

5. PROVA DE CONCEITO

A *framework* SROD foi submetida a um conjunto de testes e foram definidos um conjunto de cenários de execução ilustrado pela Figura 20. Para o correcto funcionamento da *framework* são necessários respeitar alguns pré-requisitos.

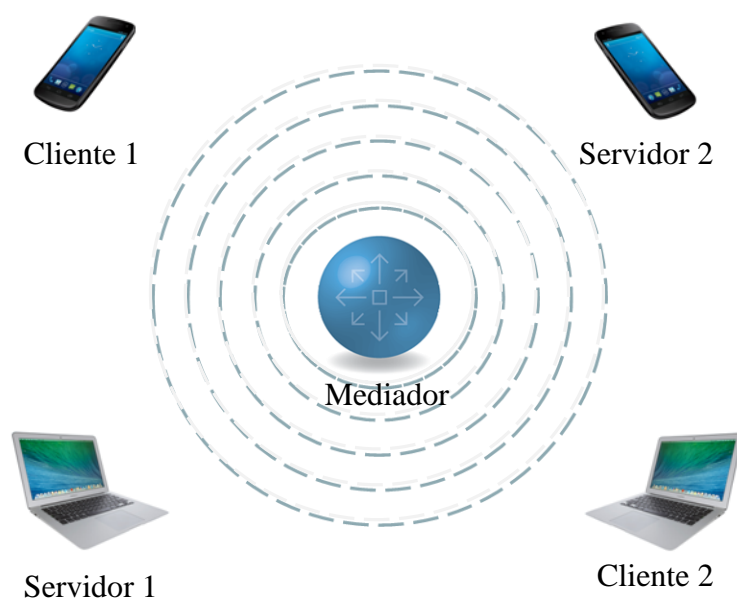


Figura 20 – Diagrama de ligações e funções desempenhadas

5.1. ESCOLHA DO MEDIADOR MQTT

Um dos componentes fulcrais da arquitectura SROD é o mediador, que essencialmente é um *broker* MQTT, existindo várias implementações disponíveis na internet. O sítio da internet que reúne as especificações do protocolo MQTT (mqtt.org), também mantém uma lista de vários *brokers* e refere dois como sendo os *brokers* a utilizar para desenvolvimento.

5.1.1. REALLY SMALL MESSAGE BROKER

O Really Small Message Broker (RSMB) é um *broker* MQTT disponibilizado livremente (mas não em código aberto) pela IBM para uso e avaliação pessoal. O RSMB está disponível na forma de um binário pré-compilado para um número limitado de plataformas.

5.1.2. MOSQUITTO MESSAGE BROKER

O Mosquitto é um *broker* MQTT v3.1 e v3.1.1, de código aberto (Licença BSD), escrito por Roger Light. O seu grande objectivo é fornecer as mesmas (e mais algumas) funcionalidades que o RSMB, mas totalmente em código aberto.

O Mosquitto é ainda recomendado para sistemas baseados em ARM V5 pelo mqtt.org, o que levou a que fosse o escolhido para ser o *broker* SROD.

5.1.3. INSTALAÇÃO DO MEDIADOR MQTT

O Mosquitto pode ser descarregado livremente a partir de <http://mosquitto.org/> quer na sua forma de código fonte quer em formato binário. Para estes testes, a versão descarregada foi em formato binário para Microsoft Windows a 32 bit.

O *broker* não foi executado como serviço, já que nesta fase é útil ler a informação de saída do *broker* na linha de comandos. Numa linha de comandos, com o seguinte comando podemos iniciar o *broker* em modo detalhado:

```
mosquitto -v
```

O *broker* em modo detalhado vai permitir a visualização de todos os tipos de registo, substituindo quaisquer opções do arquivo de configuração.

5.1.4. TESTE DE INTERACÇÃO MQTT

Junto com o *broker* mosquitto estão dois clientes básicos para linha de comandos: **mosquitto_pub** e **mosquitto_sub**. Os clientes podem ser utilizados para testes do *broker* ou combinados com uma linguagem de script para implementar funções de publicação e subscrição. Ambos os comandos serão executados com o parâmetro de depuração activo, para que possam dar o máximo de informações sobre o que está a acontecer.

- **Subscrever um tópico**

Numa linha de comandos podemos subscrever tópicos, onde o comando para subscrever mensagens do tópico de testes é:

```
mosquitto_sub -d -t test/srod
```

- **Publicar um tópico**

Numa outra linha de comandos podemos publicar uma mensagem no tópico de testes, com:

```
mosquitto_pub -d -t test/srod -m "Hello World"
```

Na linha de comandos, que está à espera de mensagens no tópico de testes, deverá aparecer o texto: “Hello World”:

O *broker* MQTT Mosquitto encontra-se configurado e a mediar comunicações no porto 1883/TCP dos vários endereços de IP da máquina onde se encontra em execução. É necessário registar o IP do *broker* para colocar nas configurações dos clientes.

Neste momento no ecossistema SROD existe um mediador funcional, de seguida é necessário garantir que o mesmo é acessível por todos os envolvidos na comunicação.

Estando o mediador em execução, foi então desenvolvida uma pequena aplicação como prova de conceito da aplicabilidade e redução de alguns erros técnicos na arquitectura da *framework*. Um protótipo foi criado para provar a viabilidade da ideia, e para mostrar não só que a *framework* funciona, mas também que ela se comporta como o previsto.

5.2. CONTEXTO

Existe uma empresa de estafetas em que o seu ramo de negócio assenta na entrega e recolha de encomendas. Os seus estafetas saem no início do dia com a sua lista de tarefas a fazer, nomeadamente entregas, mas também podem ter de efectuar algumas recolhas previamente pedidas, seguindo a sua lista de afazeres previamente criada.

Imaginando que um cliente necessita que seja recolhida na sua residência uma encomenda urgente. O cliente entrará em contacto com a linha de apoio ao cliente da empresa de estafetas que lhe vai indicar que um dos seus estafetas vai recolher a encomenda na sua residência e levar para entrega.

A linha de apoio ao cliente através do centro de distribuição irá notificar um estafeta que terá de recolher uma encomenda que não estava prevista na sua lista inicial de afazeres.

Imaginemos também, que no decorrer de uma entrega, o estafeta recebe uma encomenda que não estava contemplada para recolher na sua lista de afazeres, mas que terá de levar para o centro de distribuição, necessita, portanto, de notificar o centro da recolha extra que efectuou.

Sendo apresentado este cenário, temos então o ponto de partida para o protótipo. Deparamo-nos com a necessidade de a linha de apoio ao cliente contactar o estafeta, de modo a manipular a sua lista de afazeres para incluir a encomenda urgente na sua lista. E também da necessidade de o estafeta alterar a sua lista de afazeres notificando o centro de distribuição da sua recolha extra. Vamos, portanto, focar-nos neste ponto.

5.2.1. APLICAÇÃO SWING SROD

A *framework* recorre às funcionalidade básicas da plataforma Java SE, pelo que pode ser executada não só num ambiente móvel, mas também num ambiente Java SE. Foi então desenvolvida uma simples aplicação em java swing para mostrar as funcionalidades básicas da *framework*, de onde se mostra o ecrã principal na Figura 21.

Simplificando o problema, resume-se o mesmo a uma lista de tarefas sobre a qual tanto o cliente como o servidor de um sistema distribuído podem efectuar alterações notificando ambas as partes dessas alterações.

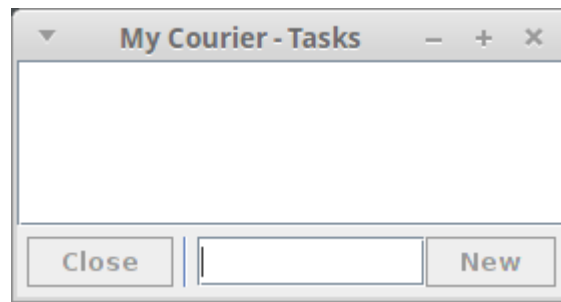


Figura 21 – Ecrã da lista de tarefas

Para manipular a lista de tarefas, e simplificando o protocolo, foi definida a interface que expõe o serviço conforme o que se apresenta no bloco de Código 6:

```
public interface CourierTasks extends Serializable {  
    boolean add(final String task);  
    boolean delete(final String task);  
    List<String> get();  
}
```

Código 6 – Interface do serviço CourierTasks

Portanto o cliente do serviço poderá adicionar tarefas, remover tarefas e listar as tarefas do fornecedor de serviço.

Do lado do fornecedor de serviço a implementação da persistência dos dados também foi simplificada e está implementada conforme o excerto de Código 7, como uma Lista, pois trata-se dum protótipo que visa testar a invocação remota de métodos, não sendo necessário grandes preocupações com a sua integridade.

```
public class CourierTasksService implements CourierTasks {  
    ...  
    private List<String> tasks = new ArrayList<String>();  
    public boolean add(String task) {  
        return tasks.add(task);  
    }  
    public boolean delete(String task) {  
        return tasks.remove(task);  
    }  
}
```

```

    public List<String> get() {

        return tasks;

    }

    ...

```

Código 7 – Implementação do serviço CourierTasksService

O cliente irá obter uma instância para o objecto remoto exposto pela interface e afectar ao seu ListModel de forma a reflectir as invocações aos métodos disponibilizados, conforme o apresentado no excerto de Código 8.

```

...
listModel = new DefaultListModel <String>();
list = new JList<String>(listModel);
...
if (courier.delete(name) ){
    listModel.remove(index);
}
...

```

Código 8 – bloco de código que demonstra a utilização do objecto remoto

Para testar a *framework* é necessário agora disponibilizar e invocar o objecto remoto. Do lado do servidor, objecto irá ser disponibilizado recorrendo ao excerto de Código 9:

```

ServiceProvider.register( Constants.SERVICE,
                           new CourierTasksService());
new MQTTInitialContext ( Constants.BROKER,
                          Constants.NAMESPACE,
                          new JSONMarshalling()
                          )
    .getProtocol()
    .startServer(Constants.SERVICE);

```

Código 9 – Instanciação e disponibilização do serviço CourierTasks

A primeira instrução regista o serviço no disponibilizador de serviços, enquanto que a segunda o torna visível num determinado contexto. Do lado do cliente, o serviço é acedido conforme o apresentado no excerto de Código 10 :

```

ClientLocator.addContext( new MQTTInitialContext(
                        Constants.BROKER,
                        Constants.NAMESPACE,
                        new JSONMarshalling()
                    ) ) ;

courier = ClientLocator.getService( Constants.SERVICE,
                                    CourierTasks.class ) ;

```

Código 10 – Acesso e localização do objecto remoto

A primeira linha indica o contexto sobre o qual serão localizados os objectos remotos, enquanto que a segunda pede um objecto que disponibilize uma interface CourierTasks nesse contexto.

Foi então invocado o cliente para adicionar 2 tarefas no servidor, onde a Figura 22 ilustra o resultado.

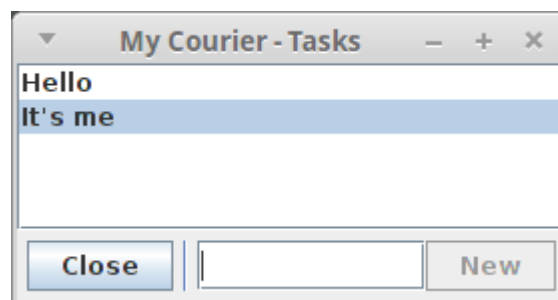


Figura 22 – Ecrã da lista de tarefas com duas tarefas adicionadas

A *framework* SROD empacotou as invocações da forma apresentada na Figura 23:

```

{"args":["Hello"],"id":1,"method":"add","service":"courier"}
  {"id":1,"value":true}
{"args":["it's me"],"id":2,"method":"add","service":"courier"}
  {"id":2,"value":true}

```

Figura 23 – Comunicações efectuadas ao invocar o objecto remoto duas vezes

Utilizando um monitorizador de protocolo do lado do servidor, podemos ter uma pequena ideia do aspecto das comunicações entre o servidor e o broker SROD, esse aspecto é ilustrado pela Figura 24.

```

00000000 10 22 00 06 4d 51 49 73 64 70 03 02 00 3c 00 14 .".MQIs dp...<..
00000010 73 65 72 67 69 6f 2e 31 34 35 31 35 31 36 30 36 sergio.1 45151606
00000020 36 36 38 34 6684
00000000 20 02 00 00 ...
00000024 82 1b 00 01 00 16 2f 73 72 6f 64 2f 63 6f 75 72 ...../s rod/cour
00000034 69 65 72 2f 71 75 65 73 74 69 6f 6e 01 ier/ques tion.
00000004 90 03 00 01 01 ....
00000009 32 56 00 16 2f 73 72 6f 64 2f 63 6f 75 72 69 65 2V../sro d/courie
00000019 72 2f 71 75 65 73 74 69 6f 6e 00 01 7b 22 61 72 r/questi on..{"ar
00000029 67 73 22 3a 5b 22 48 65 6c 6c 6f 22 5d 2c 22 69 gs":["He llo"],"i
00000039 64 22 3a 31 2c 22 6d 65 74 68 6f 64 22 3a 22 61 d":1,"me thod":"a
00000049 64 64 22 2c 22 73 65 72 76 69 63 65 22 3a 22 63 dd","ser vice":"c
00000059 6f 75 72 69 65 72 22 7d ourier"}
00000041 32 2f 00 16 2f 73 72 6f 64 2f 63 6f 75 72 69 65 2/../sro d/courie
00000051 72 2f 72 65 73 70 6f 6e 73 65 00 02 7b 22 69 64 r/respon se..{"id
00000061 22 3a 31 2c 22 76 61 6c 75 65 22 3a 74 72 75 65 ":1,"val ue":true
00000071 7d }
00000061 40 02 00 02 @...
00000072 40 02 00 01 @...
00000065 32 58 00 16 2f 73 72 6f 64 2f 63 6f 75 72 69 65 2X../sro d/courie
00000075 72 2f 71 75 65 73 74 69 6f 6e 00 02 7b 22 61 72 r/questi on..{"ar
00000085 67 73 22 3a 5b 22 49 74 27 73 20 6d 65 22 5d 2c gs":["It 's me"],
00000095 22 69 64 22 3a 32 2c 22 6d 65 74 68 6f 64 22 3a "id":2," method":
000000A5 22 61 64 64 22 2c 22 73 65 72 76 69 63 65 22 3a "add","s ervice":
000000B5 22 63 6f 75 72 69 65 72 22 7d "courier "}
00000076 32 2f 00 16 2f 73 72 6f 64 2f 63 6f 75 72 69 65 2/../sro d/courie
00000086 72 2f 72 65 73 70 6f 6e 73 65 00 03 7b 22 69 64 r/respon se..{"id
00000096 22 3a 32 2c 22 76 61 6c 75 65 22 3a 74 72 75 65 ":2,"val ue":true
000000A6 7d }
000000BF 40 02 00 03 @...
000000A7 40 02 00 02 @...

```

Figura 24 – Comunicações entre o servidor e o mediador SROD

Analisando o registo das comunicações, podemos validar o fluxo de eventos:

- O servidor regista-se no mediador SROD, que lhe confirma o registo.
- De seguida o servidor indica que quer receber notificações relativas ao tópico do serviço
- Quando um pedido chega ao mediador, o mediador invoca o serviço com o pedido do cliente ao qual o servidor entrega a resposta da invocação para ser de seguida entregue ao cliente.
- O mediador de seguida entrega a resposta ao cliente e notifica o servidor que a mesma foi entregue.

Algo que pode ser resumido, pelo diagrama apresentado na Figura 25:

```
[SROD]: Messages Subscribed!
      {"args":["Hello"],"id":1,"method":"add","service":"courier"}
      add: Hello
      {"id":1,"value":true}
[SROD]: [/srod/courier/response] Response published!
[SROD]: [/srod/courier/response] Response delivered!
      {"args":["it's me"],"id":2,"method":"add","service":"courier"}
      add: it's me
      {"id":2,"value":true}
[SROD]: [/srod/courier/response] Response published!
[SROD]: [/srod/courier/response] Response delivered!
```

Figura 25 – Detalhe das comunicações efectuadas ao invocar o objecto remoto

Resultando do lado do servidor o output para a consola apresentado na Figura 26.

```
add: Hello
add: it's me
```

Figura 26 – *Output* apresentado na consola

Estando o cliente a correr num ambiente Java SE, foi desenvolvido um segundo protótipo recorrendo à *framework* RMI da plataforma Java, como forma de comparação de *frameworks*.

5.2.2. APLICAÇÃO SWING RMI

Para manipular a lista de tarefas foi definida uma Interface, descrita no bloco de Código 11, que expõe o serviço, da mesma forma que em SROD, mas neste caso a interface estende de Remote e os seus métodos tem de declarar que podem enviar a exceção RemoteException:

```
public interface CourierTasks extends Remote {  
    boolean add(final String task) throws RemoteException;  
    boolean delete(final String task) throws RemoteException;  
    List<String> get() throws RemoteException;  
}
```

Código 11 – Exemplo de interface remota em RMI

Portanto, de igual modo o cliente do serviço poderá adicionar tarefas, remover tarefas e listar as tarefas do fornecedor de serviço.

Do lado do fornecedor de serviço a implementação da persistência dos dados não tem grandes diferenças, conforme o apresentado no excerto de Código 12, com a exceção de ter de estender a classe UnicastRemoteObject

```
public class CourierTasksService extends UnicastRemoteObject  
    implements CourierTasks {  
    ...  
    private List<String> tasks = new ArrayList<String>();  
    public boolean add(String task) {  
        return tasks.add(task);  
    }  
  
    public boolean delete(String task) {  
        return tasks.remove(task);  
    }  
  
    public List<String> get() {  
        return tasks;  
    }  
    ...  
}
```

Código 12- Exemplo de implementação da interface remota

O cliente irá obter uma instância para o objecto remoto exposto pela interface e afectar ao seu ListModel de forma a reflectir as invocações aos métodos disponibilizados. Para testar a aplicação desenvolvida em RMI é necessário agora disponibilizar e invocar o objecto remoto.

Do lado do servidor, objecto é disponibilizado pelo excerto de Código 13

```
Registry registry = LocateRegistry.createRegistry(Constants.PORT) ;  
registry.bind(Constants.SERVICE, new CourierTasksService());
```

Código 13 – Disponibilização do objecto remoto em RMI

A primeira linha cria um Registro de serviços. Enquanto que a segunda linha disponibiliza o serviço nesse registro. Enquanto que do lado do cliente, O objecto remoto é obtido pelo excerto de código 14.

```
Registry registry = LocateRegistry.getRegistry( Constants.SERVER,  
                                                Constants.PORT) ;  
courier =(CourierTasks) registry.lookup(Constants.SERVICE) ;
```

Código 14 – Localização e acesso ao objecto remoto em RMI

A primeira linha localiza o Registro de serviços sobre o qual serão localizados os objectos remotos, enquanto que a segunda linha localiza o objecto que disponibiliza uma interface CourierTasks. Como se pode verificar a nível de localização e disponibilização de serviços, ambas as *frameworks* são bastante similares. Foi então invocado o cliente para adicionar as mesmas 2 tarefas no servidor, o que está visível na Figura 27

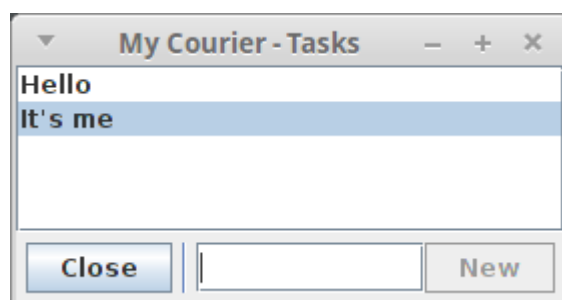


Figura 27 – Ecrã da lista de tarefas com duas tarefas adicionadas

Da mesma forma, foi utilizando um monitorizador de protocolo do lado do servidor, de modo que se tenha uma pequena ideia do aspecto das comunicações.

```

00000000 4a 52 4d 49 00 02 4b JRMI..K
00000000 4e 00 09 31 32 37 2e 30 2e 30 2e 31 00 00 d3 3a N..127.0 .0.1...:
00000007 00 09 31 32 37 2e 30 2e 31 2e 31 00 00 00 00 ..127.0. 1.1....
00000016 50 ac ed 00 05 77 22 00 00 00 00 00 00 00 02 00 P....w". ....
00000026 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000036 01 f6 b6 89 8d 8b f2 86 43 75 72 00 18 5b 4c 6a ..... Cur..[Lj
00000046 61 76 61 2e 72 6d 69 2e 73 65 72 76 65 72 2e 4f ava.rmi. server.O
00000056 62 6a 49 44 3b 87 13 00 b8 d0 2c 64 7e 02 00 00 00 bjid;... ..,d~...
00000066 70 78 70 00 00 00 01 73 72 00 15 6a 61 76 61 2e pxp....s r..java.
00000076 72 6d 69 2e 73 65 72 76 65 72 2e 4f 62 6a 49 44 rmi.serv er.ObjID
00000086 a7 5e fa 12 8d dc e5 5c 02 00 02 4a 00 06 6f 62 .^.....\ ...J..ob
00000096 6a 4e 75 6d 4c 00 05 73 70 61 63 65 74 00 15 4c jNumL..s pacet..L
000000A6 6a 61 76 61 2f 72 6d 69 2f 73 65 72 76 65 72 2f java/rmi /server/
000000B6 55 49 44 3b 70 78 70 7b 27 5e e8 58 f2 b9 19 73 UID;pxp{ '^..X...s
000000C6 72 00 13 6a 61 76 61 2e 72 6d 69 2e 73 65 72 76 r..java. rmi.serv
000000D6 65 72 2e 55 49 44 0f 12 70 0d bf 36 4f 12 02 00 er.UID.. p..60...
000000E6 03 53 00 05 63 6f 75 6e 74 4a 00 04 74 69 6d 65 .S..coun tJ..time
000000F6 49 00 06 75 6e 69 71 75 65 70 78 70 80 01 00 00 I..uniqu expx....
00000106 01 51 f4 c8 e7 bf 53 8d fc b9 77 08 80 00 00 00 .Q....S. ..w.....
00000116 00 00 00 00 73 72 00 12 6a 61 76 61 2e 72 6d 69 ....sr.. java.rmi
00000126 2e 64 67 63 2e 4c 65 61 73 65 b0 b5 e2 66 0c 4a .dgc.Lea se...f.J
00000136 dc 34 02 00 02 4a 00 05 76 61 6c 75 65 4c 00 04 .4...J.. valueL..
00000146 76 6d 69 64 74 00 13 4c 6a 61 76 61 2f 72 6d 69 vmidt..L java/rmi
00000156 2f 64 67 63 2f 56 4d 49 44 3b 70 78 70 00 00 00 /dgc/VMI D;pxp...
00000166 00 00 09 27 c0 73 72 00 11 6a 61 76 61 2e 72 6d ...'.sr. .java.rm
00000176 69 2e 64 67 63 2e 56 4d 49 44 f8 86 5b af a4 a5 i.dgc.VM ID...[...
00000186 6d b6 02 00 02 5b 00 04 61 64 64 72 74 00 02 5b m....[.. addrt..[
00000196 42 4c 00 03 75 69 64 71 00 7e 00 03 70 78 70 75 BL..uidq .~..pxpu
000001A6 72 00 02 5b 42 ac f3 17 f8 06 08 54 e0 02 00 00 r..[B... ..T....
000001B6 70 78 70 00 00 00 08 de 95 5f 4a 61 03 83 94 73 pxp..... ._Ja...s
000001C6 71 00 7e 00 05 80 01 00 00 01 51 f4 c9 2b 44 0a q.~..... ..Q...+D.
000001D6 07 cb 44 ..D
00000010 51 ac ed 00 05 77 0f 01 53 8d fc b9 00 00 01 51 Q....w.. S.....Q
00000020 f4 c8 e7 bf 80 03 73 72 00 12 6a 61 76 61 2e 72 .....sr ..java.r
00000030 6d 69 2e 64 67 63 2e 4c 65 61 73 65 b0 b5 e2 66 mi.dgc.L ease...f
00000040 0c 4a dc 34 02 00 02 4a 00 05 76 61 6c 75 65 4c .J.4...J ..valueL
00000050 00 04 76 6d 69 64 74 00 13 4c 6a 61 76 61 2f 72 ..vmidt. .Ljava/r
00000060 6d 69 2f 64 67 63 2f 56 4d 49 44 3b 70 78 70 00 mi/dgc/V MID;pxp.
00000070 00 00 00 00 09 27 c0 73 72 00 11 6a 61 76 61 2e .....'.s r..java.
00000080 72 6d 69 2e 64 67 63 2e 56 4d 49 44 f8 86 5b af rmi.dgc. VMID..[.
00000090 a4 a5 6d b6 02 00 02 5b 00 04 61 64 64 72 74 00 ..m....[ ..addrt.
000000A0 02 5b 42 4c 00 03 75 69 64 74 00 15 4c 6a 61 76 .[BL..ui dt..Ljav
000000B0 61 2f 72 6d 69 2f 73 65 72 76 65 72 2f 55 49 44 a/rmi/se rver/UID
000000C0 3b 70 78 70 75 72 00 02 5b 42 ac f3 17 f8 06 08 ;pxpur.. [B.....
000000D0 54 e0 02 00 00 70 78 70 00 00 00 08 de 95 5f 4a T....pxp ....._J
000000E0 61 03 83 94 73 72 00 13 6a 61 76 61 2e 72 6d 69 a...sr.. java.rmi
000000F0 2e 73 65 72 76 65 72 2e 55 49 44 0f 12 70 0d bf .server. UID..p..
00000100 36 4f 12 02 00 03 53 00 05 63 6f 75 6e 74 4a 00 6O....S. .countJ.
00000110 04 74 69 6d 65 49 00 06 75 6e 69 71 75 65 70 78 .timeI.. uniquepx
00000120 70 80 01 00 00 01 51 f4 c9 2b 44 0a 07 cb 44 p.....Q. .+D...D
000001D9 52 R
0000012F 53 S
000001DA 50 ac ed 00 05 77 22 7b 27 5e e8 58 f2 b9 19 53 P....w"{ '^..X...S
000001EA 8d fc b9 00 00 01 51 f4 c8 e7 bf 80 01 ff ff ff .....Q. ....
000001FA ff b8 cf 1d 55 95 1d a5 13 74 00 05 48 65 6c 6c ....U... .t..Hell
0000020A 6f o
00000130 51 ac ed 00 05 77 10 01 53 8d fc b9 00 00 01 51 Q....w.. S.....Q
00000140 f4 c8 e7 bf 80 04 01 .....
0000020B 52 R
00000147 53 S
0000020C 50 ac ed 00 05 77 22 7b 27 5e e8 58 f2 b9 19 53 P....w"{ '^..X...S
0000021C 8d fc b9 00 00 01 51 f4 c8 e7 bf 80 01 ff ff ff .....Q. ....
0000022C ff b8 cf 1d 55 95 1d a5 13 74 00 07 49 74 27 73 ....U... .t..It's
0000023C 20 6d 65 me
00000148 51 ac ed 00 05 77 10 01 53 8d fc b9 00 00 01 51 Q....w.. S.....Q
00000158 f4 c8 e7 bf 80 05 01 .....

```

Figura 28 – Comunicações RMI

Como o que se pode observar na Figura 28, a comunicação RMI, é baseada num formato binário, não é para ser interpretada por humanos, mas sim optimizada para máquinas.

Pode-se observar ainda, que existe um grande *overhead* no estabelecimento da ligação e de seguida acontece a invocação do método, pois nota-se a String que foi passada por parâmetro a circular na comunicação.

O RMI oferece um registo básico das chamadas, apenas definindo a propriedade java.java.rmi.server.logCalls como true, e por exemplo, iniciando a classe servidor com:

```
java -Djava.rmi.server.logCalls=true <ServerClass> ...
```

O RMI também permite instalar uma fábrica customizada de *sockets* RMI. Esta customização é disponibilizada para que o protocolo RMI se abstraia dos detalhes reais da comunicação, permitindo aos *sockets* serem substituídos por alternativas, tais como comunicações não *socket* ou transferência de dados codificados ou comprimidos [90].

Tendo isto em consideração, é de notar que a manipulação da *framework* RMI está fora deste âmbito.

5.2.3. APLICAÇÃO ANDROID SROD

Foi desenvolvida também uma aplicação Android, como forma de provar o conceito da *framework* srod. A aplicação foi desenvolvida para a versão 23 (6.0 - Marshmallow) do Android com retro compatibilidade com a versão 15 (4.0.3 - Ice Cream Sandwich), o que na prática significa que irá abranger 97.3% dos dispositivos actualmente activos na Google Play Store [91]

Nas aplicações Android é criada automaticamente uma *thread* principal (também designada de UI *thread*). Essa thread é muito importante porque está encarregada de encaminhar os vários eventos (incluindo os eventos de desenho) para os respectivos *widgets*. Ela é também o meio utilizado para interagir com os *widgets* do Android. Por exemplo, ao tocar num botão no ecrã, a thread UI encaminha o evento de toque para o *widget* que por sua vez define seu estado pressionado enviando de seguida um pedido para a fila de eventos solicitando que seja desenhado novamente. A UI *Thread* retira o pedido da fila de eventos e notifica o *widget* para se redesenhar. Este modelo de thread único pode produzir um mau resultado quando não se consideram as implicações. Uma

vez que tudo acontece numa única *thread*, ao serem executadas operações longas, tais como acesso à rede ou consultas a base de dados, toda a interface de utilizador será bloqueada. Nenhum dos eventos será encaminhado, incluindo os eventos de desenho, enquanto a operação longa está em execução. Da perspectiva do utilizador a aplicação parece encontrar-se “pendurada”. E quando a UI thread fica bloqueada por mais de alguns segundos é apresentado ao utilizador uma caixa de diálogo com a mensagem "A aplicação não está a responder" (ANR) [92].

A aplicação Android não foi desenhada segundo estas boas práticas, e as chamadas aos objectos remotos estão a ser feitas dentro da thread de UI, pelo que, se a aplicação provocar um uso intensivo da CPU aumenta-se o risco de erros ANR. Desta forma foi simulada uma má utilização da *framework*, para se perceber as consequências da utilização dos objectos remotos.

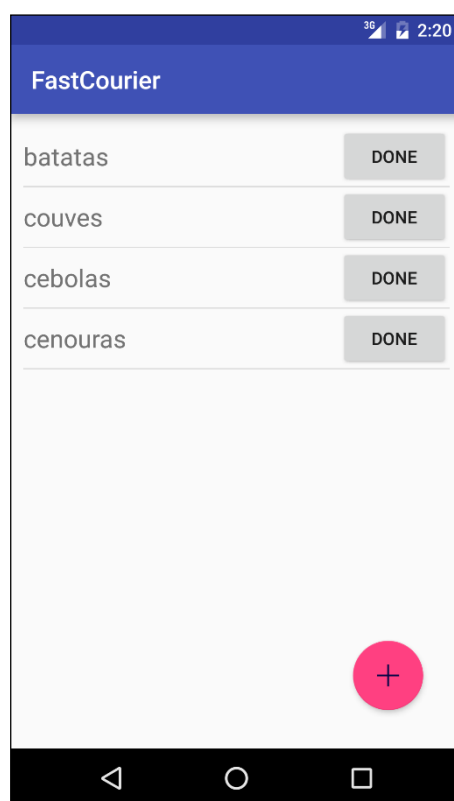


Figura 29 – Aplicação Android a correr no emulador

A aplicação apresentada pela Figura 29, inicia-se com um conjunto de valores pré preenchidos como forma de teste. A aplicação consiste numa lista de itens que podem ser marcados como concluídos e removidos da lista pelo botão “done” que se encontra

ao lado do respectivo item. Existe um botão de acção flutuante que permite a introdução de novas tarefas.

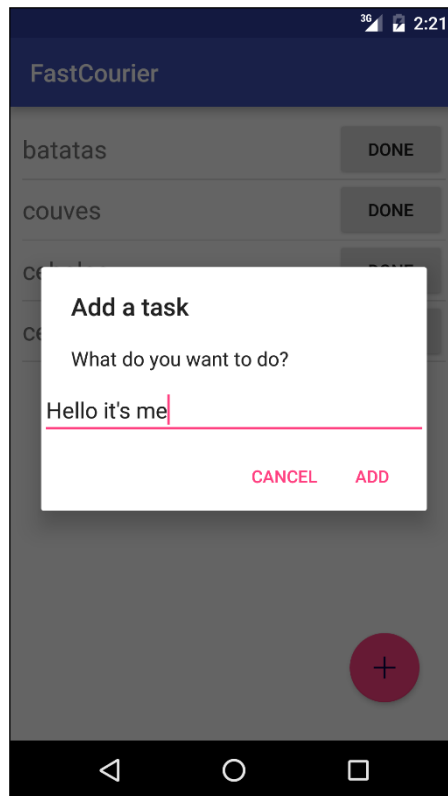


Figura 30 – Aplicação Android – Adicionar uma tarefa

A aplicação foi escrita tendo como base o componente Courier utilizado na aplicação em SWING. Dentro dos eventos dos botões, os métodos do objecto courier irão ser invocados. Tome-se como exemplo o corpo do método `onClick` do `OnClickListener` associado ao `FloatingActionButton`, que consiste na criação e apresentação de uma caixa de diálogo, conforme o demonstrado na Figura 30, a questionar o utilizador pela tarefa a adicionar a lista.

O bloco de código necessário para implementar este comportamento é apresentado pelo excerto de Código 15

```

private Courier courier;

...
new AlertDialog.Builder(MainActivity.this)
.setTitle("Add a task")
.setMessage("What do you want to do?")
.setView(inputField)
.setPositiveButton("Add", new DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialogInterface, int i) {
        final String task = inputField.getText().toString();
        if (MainActivity.this.courier.add(task)) {
            new DBHelper(MainActivity.this).addTask(task);
            MainActivity.this.listAdapter.getCursor().requery();
        }
        updateUI();
    }
})
.setNegativeButton("Cancel", null)
.create()
.show();

```

Código 15 – Implementação de uma caixa de diálogo a questionar o utilizador

Quando o utilizador escreve a tarefa e clica no botão Positivo (neste caso o “Add”), será invocado o método add do objecto remoto courier. E se o método add retornar verdadeiro, indicando a correcta adição do valor remotamente, então o valor é colocado na base de dados local e é pedido uma actualização dos dados a serem apresentados na lista.

A Figura 31 mostra o resultado da adição da tarefa á lista de tarefas da aplicação Android.

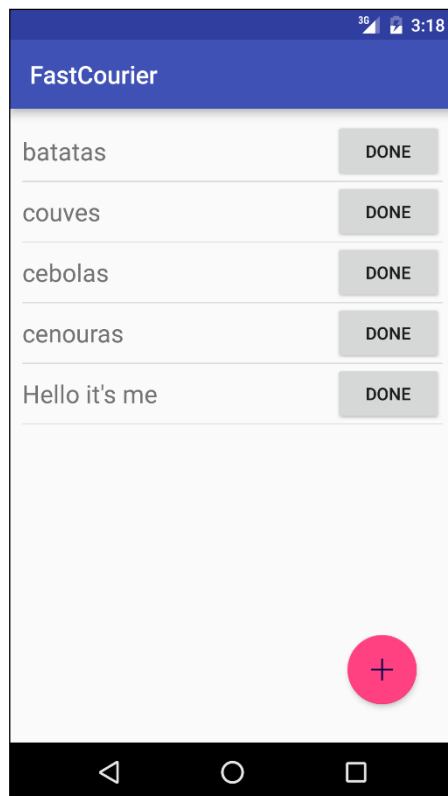


Figura 31 – Aplicação Android – Tarefa adicionada

6. CONCLUSÕES

A necessidade de uma uniformização na comunicação entre diversos tipos de dispositivos e a fragmentação provocada pela diversidade de sistemas operativos móveis com todas as suas diferentes versões levou á necessidade da criação de uma *framework* que ajudasse no desenvolvimento de aplicações móveis. Foram analisados conceitos, protocolos e sistemas operativos com vista a analisar e procurar uma maior compatibilidade dentro do ecossistema das aplicações. Foi então desenvolvida uma *framework* de comunicação transparente para o utilizador capaz de responder as várias particularidades do ambiente multiplataforma e multiequipamento. Avaliou-se o seu desempenho comparativamente a uma *framework* de referência que não tem suporte para dispositivos móveis.

Muito foi escrito sobre plataformas, sistemas operativos e protocolos, tentou-se encontrar um protocolo que fosse capaz de responder a todos os requisitos, quando não se encontrou foram interligados protocolos para formarem uma *framework*. Foi analisado profundamente o ecossistema Android, sendo que pouco das suas características influenciaram o desenvolvimento da *framework* apenas se recorreu apenas ao básico da plataforma java disponibilizado tanto pelo Android tal como pela plataforma Java SE.

O facto da plataforma Android disponibilizar as classes java através do Apache Harmony faz com que nem todos os pacotes da plataforma Java sejam disponibilizados, limitando

bastante o que se consegue atingir. Caso o Android disponibilizasse o pacote do RMI, este seria uma boa aposta como *framework* de invocação remota de métodos num âmbito das plataformas Java. As técnicas de introspecção utilizadas tiveram de ser revistas já que algumas classes utilitárias também não eram disponibilizadas.

Poder-se-ia ter escrito muito mais sobre a *framework*, apesar de se ter tentado abordar decisões de arquitectura e isolamento, boas práticas em termos de arquitectura de *software*.

Foram feitos poucos testes para garantir a sua estabilidade e integridade. Qualquer conjunto de *software* tem de ter uma grande bateria de testes associada, só desta forma pode-se garantir o correcto funcionamento e a não regressão em versões futuras do mesmo.

A *framework* foi pensada para interagir com o maior número de plataformas possíveis, utilizando tecnologias de comunicação entre máquinas utilizadas na Internet das coisas, pelo que através desta *framework* conseguimos activar um temporizador ou um termostato desde que estes disponibilizem uma interface utilizando JSON sobre o protocolo MQTT.

Existe bastante espaço para evoluir este trabalho. Desde tirar partido de todas as funcionalidades disponibilizadas pelos protocolos, passando por optimizações da *framework* e melhorias técnicas no âmbito da serialização JSON. A *framework* pode evoluir em termos de segurança das comunicações e compressão de dados a transmitir, sempre balanceando o custo de compressão com o custo energético.

Obteve-se uma *framework* de utilização simples e de fácil análise protocolar, já que recorrendo a um analisador de rede se conseguiu observar e perceber o protocolo. Sendo um protocolo de simples compreensão por humanos não é necessariamente um protocolo de simples compreensão por parte dos equipamentos electrónicos.

Foi clonado o código fonte da *framework* e das aplicações de teste para um repositório de acesso público no GitHub (<https://github.com/sergio-rodrigues>) como forma de disponibilizar a *framework* para que seja facilitado a sua evolução por parte da comunidade open-source.

No final deste trabalho conseguiu-se uma *framework* que irá facilitar o desenvolvimento de aplicações, tanto para ambientes com dispositivos móveis, mas também para aplicações em ambientes de computação convencional ou até mesmo *cloud computing*.

Referências Documentais

- [1] M. Weiser, “Ubiquitous Computing,” 17 03 1996. [Online]. Available: <http://www.ubiq.com/hypertext/weiser/UbiHome.html>.
- [2] M. Weiser, “Ubiquitous Computing,” 16 08 1996. [Online]. Available: <http://www.ubiq.com/hypertext/weiser/UbiCompHotTopics.html>.
- [3] ppinto, “Apresentação,” 08 10 2015. [Online]. Available: <http://mcm.ipg.pt/doku.php>.
- [4] M. Weiser, “The Computer for the 21st Century,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, nº 3, pp. 3-11, 1999.
- [5] Lopez Research LLC, “An Introduction to the Internet of Things,” 11 2013. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf.
- [6] Msdn.microsoft.com, “ Chapter 1: Service Oriented Architecture (SOA),” Microsoft, [Online]. Available: <https://msdn.microsoft.com/en-us/library/bb833022.aspx>.
- [7] “Service-Oriented Integration,” msdn.microsoft.com, [Online]. Available: <https://msdn.microsoft.com/en-us/library/ms978594.aspx>.
- [8] A. S. W. Andrew S Tanenbaum, Operating Systems Design and Implementation, Prentice Hall, 2006.
- [9] D. K. B. Coulouris, “Distributed Systems | Concepts and Design, Fifth Edition,” [Online]. Available: <http://www.cdk5.net/wp/>. [Acedido em 1 1 2014].
- [10] R. Toal, “Remoting,” 03 2002. [Online]. Available: <http://cs.lmu.edu/~ray/notes/remoting/>.

- [11] Q. H. Mahmoud, “Sockets programming in Java: A tutorial,” 11 12 1996. [Online]. Available: <http://www.javaworld.com/article/2077322/core-java/core-java-sockets-programming-in-java-a-tutorial.html>.
- [12] R. Thurlow, “RPC: Remote Procedure Call Protocol Specification Version 2,” 05 2009. [Online]. Available: <https://tools.ietf.org/html/rfc5531>.
- [13] Oracle, “An Overview of RMI Applications,” 01 1995. [Online]. Available: <https://docs.oracle.com/javase/tutorial/rmi/overview.html>.
- [14] W3C, “SOAP Version 1.2 Part 2: Adjuncts (Second Edition),” 27 04 2007. [Online]. Available: <https://www.w3.org/TR/2007/REC-soap12-part2-20070427/>.
- [15] T. Gao, “Chapter 13 .NET Remoting,” em *The Complete Reference to Professional Soa with Visual Studio 2005 (C# & VB 2005) .Net 3.0*, Lulu.com, 2007, p. 418.
- [16] Object Management Group, “CORBA® 3.3,” 11 2012. [Online]. Available: <http://www.omg.org/spec/CORBA/Current/>.
- [17] D. Winer, “XML-RPC Specification,” 7 1999. [Online]. Available: <http://xmlrpc.scripting.com/spec.html>.
- [18] JSON-RPC.ORG, “JSON-RPC 1.0 Specifications ¶,” 07 08 2012. [Online]. Available: <http://json-rpc.org/wiki/specification>.
- [19] ietf, “Hypertext Transfer Protocol -- HTTP/1.0,” 05 1996. [Online]. Available: <https://tools.ietf.org/html/rfc1945>.
- [20] H. & T. & M. & A. & K. & Boyle, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*, 1st Edition, Academic Press, 2014.
- [21] J. Rebes, *The Dzone Guide to the Internet of Things*, Dzone, 2015.
- [22] Oracle, “JSR 343: Java™ Message Service 2.0,” 21 05 2013. [Online]. Available: <https://jcp.org/en/jsr/detail?id=343>.

- [23] OASIS, “OASIS Advanced Message Queuing Protocol (AMQP),” 21 01 2012. [Online]. Available: <https://www.oasis-open.org/news/announcements/pr-30-day-public-review-for-oasis-amqp-version-1-0>.
- [24] andyp, “MQTT for Sensor Networks – MQTT-SN,” 02 12 2013. [Online]. Available: <http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn>.
- [25] International Business Machines Corporation (IBM), “MQTT V3.1 Protocol Specification,” 2010. [Online]. Available: <https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>.
- [26] R. T. Fielding, “Representational State Transfer (REST),” 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
- [27] “The Constrained Application Protocol (CoAP),” 06 2014. [Online]. Available: <https://tools.ietf.org/html/rfc7252>.
- [28] Object Management Group, “Data Distribution Service™, V1.2,” 01 2007. [Online]. Available: <http://www.omg.org/spec/DDS/1.4/>.
- [29] C. Welch, “Before it took over smartphones, Android was originally destined for cameras,” 16 04 2013. [Online]. Available: <http://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones>.
- [30] Open Handset Alliance, “Industry Leaders Announce Open Platform for Mobile Devices,” 05 11 2007. [Online]. Available: http://www.openhandsetalliance.com/press_110507.html.
- [31] Blackberry, “An update for BlackBerry® 10 Developers,” 26 10 2015. [Online]. Available: <http://devblog.blackberry.com/2015/10/an-update-for-blackberry-10-developers/>.
- [32] Apple, “Apple Developer,” 01 2013. [Online]. Available: <https://developer.apple.com/>.
- [33] C. Ziegler, “Microsoft talks Windows Phone 7 Series development ahead of GDC: Silverlight, XNA, and no backward compatibility,” 03 04 2010. [Online]. Available:

<http://www.engadget.com/2010/03/04/microsoft-talks-windows-phone-7-series-development-ahead-of-gdc/>.

- [34] D. Treadwell, “Expanding the Universal Windows Platform at Build 2015,” 29 04 2015. [Online]. Available: <https://blogs.windows.com/buildingapps/2015/04/29/expanding-the-universal-windows-platform-at-build-2015/>.
- [35] mozilla.org, “Firefox OS,” [Online]. Available: <https://www.mozilla.org/en-US/firefox/os/>.
- [36] MDN, “Firefox OS - Mozilla | MDN,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Firefox_OS.
- [37] MDN, “Firefox OS architecture - Mozilla | MDN,” [Online]. Available: https://developer.mozilla.org/en-US/Firefox_OS/Platform/Architecture.
- [38] sailfishos.org, “About - sailfishos.org,” [Online]. Available: <https://sailfishos.org/about/>.
- [39] Myriad, “Myriad Powers Android Apps on First Jolla Smartphone,” 20 11 2013. [Online]. Available: <http://www.myriadgroup.com/en/press/news/myriad-powers-android-apps-on-first-jolla-smartphone/>.
- [40] sailfishos.org, “Software Development Kit - sailfishos.org,” [Online]. Available: <https://sailfishos.org/develop/sdk-overview/>.
- [41] tizen.org, “About | Tizen,” [Online]. Available: <https://www.tizen.org/about>.
- [42] A. Saxena, “Samsung to finally merge Bada with Tizen, might announce a Tizen phone at the MWC,” 26 02 2013. [Online]. Available: <http://gadgets.ndtv.com/mobiles/news/samsung-to-finally-merge-bada-with-tizen-might-announce-a-tizen-phone-at-the-mwc-335587>.
- [43] tizen.org, “Tizen Web UI Practices,” [Online]. Available: <https://developer.tizen.org/development/ui-practices/web-application>.
- [44] M. Brian, “Samsung's NX300M smart camera is its first to run Tizen OS,” 11 11 2013. [Online]. Available: <http://www.engadget.com/2013/11/11/samsungs-nx300m-mirrorless-camera-is-its-first-to-run-tizen-os/>.

- [45] M. Shuttleworth, “Ubuntu on phones, tablets, TV’s and smart screens everywhere,” 31 10 2011. [Online]. Available: <http://www.markshuttleworth.com/archives/820>.
- [46] Canonical , “Canonical | Open source projects directory,” [Online]. Available: <http://www.canonical.com/projects/directory>.
- [47] Banco Mundial, “Pessoas com acesso à Internet por 100 habitantes - Dados de Banco Mundial,” 20 08 2014. [Online]. Available: <https://www.google.com/publicdata/>.
- [48] Grupo Marktest, “4 milhões com smartphone,” 02 09 2014. [Online]. Available: <http://www.marktest.com/wap/a/n/id~1dba.aspx>.
- [49] Gartner, “Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments to Grow 4.2 Percent in 2014,” 07 07 2014. [Online]. Available: <https://www.gartner.com/newsroom/id/2791017>.
- [50] Open Handset Alliance, “Android FAQ | Open Handset Alliance,” [Online]. Available: http://www.openhandsetalliance.com/android_overview.html.
- [51] android, “SDK Tools Release Notes | Android Developers,” [Online]. Available: <https://developer.android.com/tools/sdk/tools-notes.html>.
- [52] Android, “Android Interfaces and Architecture,” [Online]. Available: <https://source.android.com/devices/>.
- [53] Android, “The Android Source Code | Android Open Source Project,” [Online]. Available: <https://source.android.com/source/index.html>.
- [54] Google, “kernel/ - Git at Google,” [Online]. Available: <https://android.googlesource.com/kernel/>.
- [55] Android, “Codenames, Tags, and Build Numbers | Android Open Source Project,” [Online]. Available: <https://source.android.com/source/build-numbers.html>.
- [56] Android, “Managing Your App's Memory | Android Developers,” [Online]. Available: <https://developer.android.com/training/articles/memory.html>.

- [57] Android, “Security Tips | Android Developers,” [Online]. Available: <https://developer.android.com/training/articles/security-tips.html>.
- [58] Android, “Keeping the Device Awake | Android Developers,” [Online]. Available: <https://developer.android.com/training/scheduling/wakelock.html>.
- [59] M. T. Jones, “Introducing the 3.3 and 3.4 Linux kernels,” 19 06 2012. [Online]. Available: <https://www.ibm.com/developerworks/library/l-33linuxkernel/>.
- [60] M. T. Jones, “Anatomy of Linux dynamic libraries,” 20 08 2008. [Online]. Available: <https://www.ibm.com/developerworks/library/l-dynamic-libraries/>.
- [61] J. W. Buse, “Linux Network Stack,” 17 09 2013. [Online]. Available: <http://www.linux.org/threads/linux-network-stack.4620/>.
- [62] J. Huang, “Android IPC Mechanism,” 19 03 2012. [Online]. Available: <https://www.dre.vanderbilt.edu/~schmidt/cs282/PDFs/android-binder-ipc.pdf>.
- [63] A. Gargenta, “Deep Dive into Android IPC/Binder Framework at Android Builders Summit 2013,” 19 02 2013. [Online]. Available: http://events.linuxfoundation.org/images/stories/slides/abs2013_gargentas.pdf.
- [64] Android, “Media | Android Open Source Project,” [Online]. Available: <https://source.android.com/devices/media/>.
- [65] Android, “Android NDK | Android Developers,” [Online]. Available: <https://developer.android.com/ndk/index.html>.
- [66] Android, “Graphics | Android Open Source Project,” [Online]. Available: <https://source.android.com/devices/graphics/index.html>.
- [67] SQLite.org, “About SQLite,” [Online]. Available: <https://www.sqlite.org/about.html>.
- [68] Khronos Group, “OpenGL ES - The Standard for Embedded Accelerated 3D Graphics,” [Online]. Available: <https://www.khronos.org/opengles/>.

- [69] W. Lemberg, “The FreeType Project,” [Online]. Available: <http://www.freetype.org/>.
- [70] Apple Inc., “WebKit,” [Online]. Available: <https://webkit.org/>.
- [71] skia.org, “Docs,” [Online]. Available: <https://skia.org/>.
- [72] Android, “Security with HTTPS and SSL | Android Developers,” [Online]. Available: <https://developer.android.com/training/articles/security-ssl.html>.
- [73] D. Lazăr, “Why did Google rewrite libc, creating Bionic C library?,” 24 11 2013 . [Online]. Available: <https://www.quora.com/Why-did-Google-rewrite-libc-creating-Bionic-C-library>.
- [74] msdn.microsoft, “Compiled vs. Interpreted Applications,” [Online]. Available: [https://msdn.microsoft.com/en-us/library/aa240840\(v=vs.60\).aspx](https://msdn.microsoft.com/en-us/library/aa240840(v=vs.60).aspx).
- [75] Android, “Glossary | Android Developers,” [Online]. Available: <https://developer.android.com/guide/appendix/glossary.html>.
- [76] D. Bornstein, “Dalvik VM Internals,” 2008. [Online]. Available: <https://sites.google.com/site/io/dalvik-vm-internals>.
- [77] S. Buckley, “ART' experiment in Android KitKat improves battery life and speeds up apps,” 06 11 2013. [Online]. Available: <http://www.engadget.com/2013/11/06/new-android-runtime-could-improve-battery-life/>.
- [78] Android, “ART and Dalvik | Android Open Source Project,” [Online]. Available: <https://source.android.com/devices/tech/dalvik/>.
- [79] B. Burd, “ART: The New Android Runtime,” 04 07 2014. [Online]. Available: <http://www.infoq.com/news/2014/07/art-runtime>.
- [80] Android, “ActivityManager | Android Developers,” [Online]. Available: <https://developer.android.com/reference/android/app/ActivityManager.html>.
- [81] Android, “Content Providers | Android Developers,” [Online]. Available: <https://developer.android.com/guide/topics/providers/content-providers.html>.

- [82] Android, “TelephonyManager | Android Developers,” [Online]. Available: <https://developer.android.com/reference/android/telephony/TelephonyManager.html>.
- [83] Android, “LocationManager | Android Developers,” [Online]. Available: <https://developer.android.com/reference/android/location/LocationManager.html>.
- [84] Android, “Android Debug Bridge | Android Developers,” [Online]. Available: <https://developer.android.com/tools/help/adb.html>.
- [85] Android, “Android Developer Tools | Android Developers,” [Online]. Available: <https://developer.android.com/tools/help/adt.html>.
- [86] Android, “Android Studio Overview | Android Developers,” [Online]. Available: <https://developer.android.com/tools/studio/index.html>.
- [87] Android, “Building and Running Overview | Android Developers,” [Online]. Available: <https://developer.android.com/tools/building/index.html>.
- [88] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” 07 2006. [Online]. Available: <https://tools.ietf.org/html/rfc4627>.
- [89] J.-R. W. Group, “JSON-RPC 2.0 Specification,” 4 1 2014. [Online]. Available: <http://www.jsonrpc.org/specification>. [Acedido em 4 1 2014].
- [90] J. Shirazi, Java Performance Tuning, 2nd Edition, O'Reilly Media, 2003.
- [91] Android, “Dashboards | Android Developers,” [Online]. Available: <https://developer.android.com/about/dashboards/index.html>.
- [92] R. Guy, “Painless threading | Android Developers Blog,” 06 05 2009. [Online]. Available: <http://android-developers.blogspot.pt/2009/05/painless-threading.html>.