Agradecimentos

Gostaria de agradecer a todos os que me apoiaram durante o processo de desenvolvimento e escrita desta dissertação.

Em especial quero agradecer ao meu orientador, Professor Doutor Adérito Fernandes Marcos, por todo o apoio e pelo incentivo que me foi dando no decorrer do trabalho.

Dedico o esforço à minha mulher Fernanda e a toda a minha família e amigos.

Modelos peer-to-peer aplicados a sistemas de comunicação multimédia móveis

É sobejamente conhecido o impacto das TICs (Tecnologias de Informação e Comunicação) no dia-a-dia das sociedades modernas e industrializadas. A adesão maciça à comunicação móvel de voz, que (ainda) está a decorrer e que teve o seu início no final dos anos 80, levou a uma nova área de pesquisa aplicada nas TICs: a comunicação móvel de dados. Desde então, surgiram variadíssimos conceitos, tecnologias e *standards* de comunicação, como por exemplo, GPRS (*General Packet Radio Service*), HSCSD (*High Speed Circuit Switched Data*), *Bluetooth*, WAP (*Wireless Application Protocol*), *Wireless* LANs (*Local Area Networks*), e muitos outros. Além disso, surgem com grande frequência novos dispositivos como PDAs (*Personal Digital Assistants*), telemóveis e *smartphones*, *subnotebooks*, etc.. Embora disponham de um grande potencial de utilidade, capaz de transformar a Sociedade de Informação uma segunda vez depois da sua penetração pela *Internet*, verifica-se, no entanto, ainda uma lacuna substancial a nível de aplicações para estas tecnologias e dispositivos especialmente quando consideramos a possibilidade de aceder e partilhar informação sem qualquer tipo de barreiras. É neste âmbito que o tema desta dissertação de mestrado se insere: avaliar as reais capacidades de partilha dos dispositivos móveis, nomeadamente, utilizando a arquitectura de rede *peer-to-peer*.

Peer-to-peer models applied to mobile multimedia communication systems

It's well known nowadays impact of the ICT (Information and Communication Technologies) on the modern and industrialized societies. The full adhesion to the voice mobile communications that is still occurring and that began at the end of the 80's, led to a new research area applied to ICT: mobile data communication. Since then many concepts, technologies and standards have emerged such as GPRS (*General Packet Radio Service*), HSCSD (*High Speed Circuit Switched Data*), *Bluetooth*, WAP (*Wireless Application Protocol*), *Wireless* LANs (*Local Area Networks*), and many others. Furthermore, appear regularly new devices like PDAs (*Personal Digital Assistants*), mobile phones e *smartphones*, *subnotebooks*, etc. Despite their large potential of utility, enable of transform the Information Society a second time after its penetration through the Internet, we can observe a substantial gap concerning applications for those technologies and devices especially when we consider the possibility of access and share information with any type of barrier. This is the ambit of this dissertation: the evaluation of the real sharing capabilities of the mobile devices, mainly, using the peer-to-peer architecture.

Índice

Abreviaturas

API	Application Programming Interface
CA	Certificate Authority
CDC	Connected Device Configuration
CLDC	Connected, Limited Device Configuration
CLR	Common Language Runtime
CTS	Common Type System
DES	Data Encryption Standard
DNS	Domain Name Server
DSA	Digital Signature Algorithm
EJB	Enterprise Java Beans
ERP	Endpoint Routing Protocol
FTP	File Transfer Protocol
GAP	Generic Access Profile
GOEP	Generic Object Exchange Profile
GPRS	General Packet Radion Service
GSM	Global System for Mobile Communication
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IEEE	Institute of Electrical and Electronics Engineers, Inc
IEFT	Internet Engineering Task Force
IMPS	Instant Messaging and Presence Service
IP	Internet Protocol
ITS System	Intelligent Traffic Service System
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive
JCA	Java Cryptography Architecture

JCE	Java Cryptography Extension
JCP	Java Community Process
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
JXME	JXTA for J2ME
JXTA	Juxtapose
KVM	Kilobyte/Kuaui Virtual Machine
L2CAP	Logical Link Control and Adaption Protocol
LAN	Local Area Network
MID	Mobile Information Device
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
NAT	Network Address Translation
OBEX	Object Exchange Protocol
OTA	Over-The-Air
P2P	Peer-to-Peer
PAN	Personal Area Network
PATS	Program for Advanced Telecom Services
PBP	Pipe Binding Protocol
PDA	Personal Digital Assistant
PDAP	PDA Profile
PDP	Peer Discovery Protocol
PEP	Peer Endpoint Protocol
PIP	Peer Information Protocol
PKI	Public Key Infrastructure
PMP	Peer Membership Protocol
PRP	Peer Resolver Protocol
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RVP	Rendezvous Protocol
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol

SHA-1	Secure Hash Algorithm
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SIPPING	Session Initiation Proposal Investigation
SMS	Short Message Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPP	Serial Port Profile
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VAME	Visual Age Micro Edition
WAP	Wireless Application Protocol
WLAN	Wireless LAN
WSDL	Web Service Description Language
XML	Extensible Markup Language

Lista de Figuras

1 Introdução

Os computadores estão a mudar radicalmente. O poder de processamento e a capacidade de memória estão a duplicar a cada 18 meses, de acordo com a lei de Moore, e o tamanho a diminuir. Os cientistas prevêem que nos próximos anos, os computadores que prevalecerão serão pequenos, móveis e sem fios tal como os telefones celulares, *personal digital assistants* (PDA) e os *pocket PCs*.

A evolução transforma computadores completamente fixos em dispositivos móveis, e esta mobilidade resulta de uma mudança de ligações com fios (*hard-wired*) em ligações sem fios (*wireless*). De uma qualquer colecção, aleatoriamente construída, de dispositivos pequenos, móveis e sem fios pode-se esperar um ambiente computacional altamente heterogéneo. Isto resulta do facto de não existir um sistema operativo dominante para estes dispositivos (*embedded devices*). Existe uma variedade de sistemas operativos e processadores destinados a tarefas/objectivos específicos e que são programados em linguagens diferentes. Além disso, os dispositivos móveis movem-se constantemente, ora entrando em contacto com novos dispositivos, ora desligando-se, conforme os seus utilizadores se deslocam.

1.1 Motivação

Hoje, os dispositivos móveis sem fios são usados como uma extensão da rede existente com fios e pensados como meros clientes com ligações sem fios. Por exemplo, telefones celulares podem ser usados para enviar e receber *emails* e mensagens SMS. Mas os dispositivos têm um potencial infinito: formação de redes pessoais (*Personal Area Network* - PAN) e participação em aplicações distribuídas. As novas tecnologias irão brevemente tornar este cenário possível.

As aplicações para dispositivos móveis sem fios devem sobrepor-se à diversidade de plataformas e sistemas operativos. Por forma a obter-se uma aplicação que seja executável em todos os dispositivos, ela deve ser independente da plataforma. Isto pode ser conseguido através da linguagem de programação JAVA. JAVA não é apenas uma linguagem, é também uma plataforma de *software*, i.e., ambiente de programação que alcançou grande sucesso depois da sua introdução em 1995. A característica base do JAVA

é a de permitir esconder a complexidade dos dispositivos das aplicações. As aplicações ”contactam” com as interfaces estandardizadas do JAVA não tendo que lidar com as características especiais dos diferentes dispositivos. Como resultado disso, as aplicações são portáveis entre diferentes plataformas. Java 2 Micro Edition [J2ME] é uma versão do JAVA destinada a correr em dispositivos pequenos com memória limitada, tendo já, vários telefones celulares e PDAs, disponível suporte para J2ME.

Já existem tecnologias de baixo nível para fazer das arquitecturas distribuídas *peer-to-peer* uma realidade, como as *Wireless* LAN (WLAN) e *Bluetooth*. As tecnologias de alto nível encontram-se ainda em desenvolvimento: duas iniciativas da Sun (JXTA e Jini) suportam aplicações em ambientes móveis.

O âmbito desta tese de mestrado baseia-se na investigação das reais capacidades de conectividade *peer-to-peer* de dispositivos móveis, como telefones celulares e PDAs, usando J2ME e fazer a avaliação de tecnologias existentes como o JXTA para J2ME (JXME).

Esta tese fará uma breve introdução ao J2ME e CLDC/MIDP, APIs do JAVA para pequenos dispositivos e focará as capacidades disponibilizadas como parte destas plataformas. Uma tecnologia distribuída, JXTA, será introduzida sendo também feito um resumo sobre a mesma.

Um protótipo foi desenvolvido como parte deste trabalho. Apesar do estudo aprofundado do JXTA, este não pode ser usado no protótipo devido à presença de uma *firewall* no local de desenvolvimento, tornando o JXTA inoperacional. Assim, optou-se pela implementação de uma solução própria.

1.2 Objectivos e limitações

O objectivo deste trabalho é fazer uma avaliação das capacidades de comunicação dos dispositivos com capacidade Java. Um dos aspectos críticos do J2ME é a conectividade mas a questão é saber se as capacidades actuais podem ser usadas e possivelmente estendidas para suportar funcionalidades fora da arquitectura usual cliente/servidor.

As tecnologias relacionadas com o processamento cliente/servidor e o suporte de diferentes tipos de terminais que acedem a um servidor não serão consideradas nem discutidas. Nem o serão as tecnologias de baixo nível como o *Bluetooth* ou WLAN, nem as

tecnologias de acesso como GSM, GPRS e UMTS. Outras áreas que também não serão abrangidas em relação ao J2ME são XML, SOAP ou SyncML. XML é uma linguagem *markup* que manipula informação estruturada, SOAP é um protocolo baseado em XML/HTTP para aceder a serviços, objectos e servidores sendo independente da plataforma. SyncML é usado para sincronização de dados numa rede.

1.3 Guia de leitura

No sentido de facultar a leitura desta tese, que admitimos poderá ser algo extensa, é incluído este guia de leitura para melhor localização dos capítulos e áreas de interesse. Será feita, então, uma apresentação dos capítulos.

- **Introdução:** é o capítulo que está a ler agora. Faz a introdução do âmbito, os seus objectivos e limitações. Além disso, contém o guia de leitura.

- **Estado da arte:** o documento começa com um capítulo que faz uma introdução aos dispositivos móveis e integrados e aos paradigmas de tecnologias de rede cliente/servidor e *peer-to-peer*.

- **Capacidades de ligação em rede do J2ME:** apresenta o J2ME com relevo no CLDC e MIDP e as capacidades de conectividade usando estas tecnologias nos dias de hoje e quais as capacidades que terão no futuro.

- **JXTA:** foca o JXTA, uma arquitectura *peer-to-peer* da Sun, e o pacote JXTA para J2ME destinado aos dispositivos com capacidades J2ME.

- **Limitações no desenvolvimento de aplicações para dispositivos móveis:** são apresentadas as limitações inerentes ao desenvolvimento de aplicações para dispositivos móveis, não só a nível do *software* mas também das limitações físicas (*hardware*).

- **O Protótipo:** descreve o protótipo desenvolvido em relação ao projecto. O protótipo tem o nome **mGolfScoreCard**.

- **Avaliação do Protótipo:** descreve a metodologia usada para validação do protótipo desenvolvido e discute características alcançadas e outras que podem ser melhoradas.

- **Conclusão:** a tese é concluída neste capítulo com algumas recomendações e pensamentos sobre o futuro das tecnologias discutidas.

2 Estado da Arte

Os dispositivos integrados têm ganho grande sucesso e os analistas reconhecem que estes computadores serão parte integrante da nossa sociedade nas próximas décadas. A tecnologia tem muitas possibilidades mas também algumas limitações. Ambas irão influenciar os cenários aplicacionais concretos baseados nestes dispositivos.

Neste capítulo será feita uma breve descrição dos dispositivos integrados, suas capacidades e limitações em geral e em relação ao desenvolvimento de aplicações e tecnologias para ligações em rede. Depois será dada uma introdução a estas tecnologias, antes de serem apresentadas as arquitecturas cliente/servidor e a nova arquitectura *peer-to-peer*.

2.1 Dispositivos integrados

Os dispositivos integrados móveis são produtos de consumo pequenos baseados em microprocessadores, como telefones celulares, *pagers* e PDAs. Chamam-se dispositivos integrados porque os pequenos computadores dentro deles têm uma operação muito específica. A maior parte dos dispositivos móveis comunicam com outros dispositivos por ligações sem fios.

Quando se implementam aplicações para dispositivos integrados móveis deve-se ter em atenção as suas limitações, que se resumem a:

- Pouca memória e fraco poder de processamento
- Limitações nas possibilidades de introdução de dados. Trata-se, normalmente, de um teclado com aproximadamente 12-18 teclas
- Ecrãs pequenos
- Possibilidades conectivas limitadas
- Trabalham com baterias de capacidade limitada

As aplicações devem ter, também, em consideração questões de segurança. Durante os últimos anos, diversas iniciativas foram criadas por forma a implementar-se um ambiente de desenvolvimento de aplicações para dispositivos integrados. Muitas dessas iniciativas foram aplicadas na plataforma Palm, tal como a IBM e a sua Java Virtual Machine J9 e a VAME (Visual Age Micro Edition) IDE que trata todos os métodos C para o API Palm em Java. Outro exemplo são as soluções Waba [Waba] ou SuperWaba [SWaba].

Java 2 Micro Edition (J2ME), um ambiente genérico para aplicações Java para dispositivos integrados, surgiu como uma iniciativa da Sun em 1999. Esta plataforma fornece funcionalidades que juntam dispositivos com funções/propósitos diferentes dentro do mercado dos integrados. O J2ME tem ganho muita notoriedade nos últimos anos e muitas companhias como a Siemens, Motorola e Nokia adoptam esta plataforma nos seus telefones. Será apresentada uma descrição do J2ME no capítulo 3.

As aplicações móveis também têm muitas vantagens em contraste com as aplicações normais, como as conhecemos hoje. São capazes de ser altamente móveis, os serviços podem ser dependentes da localização e as aplicações móveis possibilitam ao utilizador estar sempre "ligado". Mas existem algumas restrições que devem ser consideradas quando se desenham aplicações distribuídas móveis.

Os dispositivos integrados móveis, como os telemóveis e PDAs operam no limite da rede e como resultado da sua natureza móvel, requerem tecnologias capazes de operar num ambiente onde alguns ou todos os nós são móveis. Hoje, as tecnologias sem fios mais comum são GSM, GPRS e WLAN. Novas tecnologias têm surgido como UMTS e *Bluetooth*. Estas tecnologias são maioritariamente usadas como tecnologias de acesso à *Internet*, mas a esperança para o futuro é a formação de uma rede por parte dos dispositivos sem dependerem de ligações à *Internet*. Por forma a criar tal ambiente, as funcionalidades devem correr de uma forma distribuída. Aqui, os nós entrarão e sairão na rede sem aviso, enquanto os utilizadores farão as mesmas exigências de conectividade e tráfego como nas redes tradicionais.

As características típicas para funções de rede, quando concentradas numa rede de computadores (dispositivos móveis e PDAs) são:

- Operações distribuídas

- Topologia dinâmica da rede
- Capacidade de ligação variável, i.e., menor estabilidade na conexão e variação na velocidade, bem como variação na cobertura da rede
- Dispositivos com capacidade limitada
- Possível conexão com alta latência

Uma vez que os dispositivos integrados têm capacidade limitada, como um processador pequeno, pouca memória e tempo de bateria, estes aspectos devem ser tomados em conta quando o dispositivo faz parte de uma rede. Como resultado, todos os algoritmos e mecanismos que implementam a funcionalidade da rede devem ser capazes de adaptar mudanças súbitas na qualidade de transmissão. A razão é a qualidade de comunicação variável nas redes sem fios, que se torna difícil garantir para os serviços que são anunciados para cada dispositivo.

2.2 Arquitectura da rede e tecnologias

O desenvolvimento de aplicações que se executem localmente nos dispositivos integrados pode ser suficientemente útil, mas muitos utilizadores exigem ligação a uma rede. Querem ter a capacidade de aceder a serviços em qualquer lugar, em qualquer momento, independentemente do dispositivo.

As tecnologias de rede tradicionais têm algumas limitações que impõem restrições nas aplicações e reduzem a aplicabilidade. Para compensar estas desvantagens e por forma a existir partilha de aplicações numa rede, foram desenvolvidas tecnologias distribuídas como CORBA, TINA e DCOM. No entanto, um novo paradigma das tecnologias distribuídas de rede tem emergido com alguma relevância. Trata-se da *peer-to-peer (P2P) network technology*.

2.2.1 Arquitectura Cliente/Servidor

Na arquitectura de rede cliente/servidor, os clientes conectam-se a um servidor usando um protocolo específico de comunicação por forma a aceder a um recurso específico. Um desses protocolos pode ser o *Hypertext Transfer Protocol* (HTTP) ou o *File Transfer Protocol* (FTP). Neste cenário, a grande fatia do esforço é assumida pelo servidor quando este tem

que processar e entregar o serviço, enquanto que os requisitos do cliente são pequenos. Esta arquitectura é usada pelas mais populares aplicações para a *Internet*, como a *Web* e o *Email*. A arquitectura cliente/servidor é apresentada na Figura 2.1.

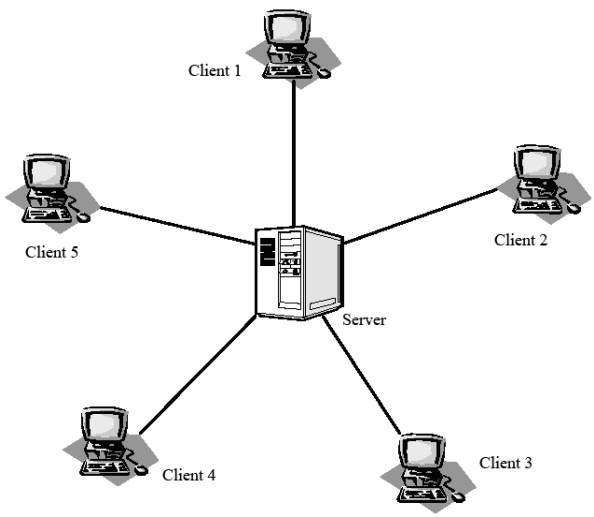


Figura 2-1 Arquitectura Cliente/Servidor

A razão para a generalização desta estratégia cliente/servidor deveu-se à inexistência de poder computacional. O equipamento era mais dispendioso e não se podia esperar que os clientes possuíssem grande poder computacional. Por outro lado, este modelo foi desenvolvido numa altura em que a maior parte das máquinas com acesso à *Internet* tinham endereços IP estáticos, i.e., era possível encontrar uma máquina usando um simples nome que era traduzido num endereço IP empregando um DNS.

Hoje em dia, podemos observar que esta arquitectura tem várias vantagens e algumas desvantagens relevantes. As vantagens são:

- **Simplicidade:** A vantagem principal dos sistemas centralizados é a sua simplicidade.

- **Maneabilidade e a coerência na informação:** Todos os dados são concentrados num só lugar e como resultado, são fáceis de gerir e não se colocam questões de consistência e coerência dos dados.

- **Segurança:** Os sistemas centralizados são relativamente fáceis de se tornarem seguros uma vez que só existe um servidor que necessita ser protegido.

As desvantagens relevantes são:

- **Escalabilidade:** Teoricamente, uma vez que o número de clientes está a aumentar, a procura de carregamento e largura de banda no servidor também está a aumentar. Isto eventualmente impede o servidor de tratar clientes adicionais. Na prática, um servidor pode ter recursos suficientes para satisfazer os pedidos dos seus utilizadores como resultado do crescimento da capacidade da memória e CPU. [Minar2002]

- **Largura de banda:** A capacidade da largura de banda tem crescido por um factor de 10^6 desde 1975, de acordo com a lei de Moore. Mas grande parte da largura de banda não é usada se toda a gente aceder apenas aos mesmos *sites* para obter os seus serviços.

- **Poder computacional:** Muitos têm investido em poder computacional que ultrapassa o poder de processamento necessário para usar a maior parte das aplicações da *Internet,* tais como navegar na *web* e ler/enviar mensagens de *email*. Noutras palavras, existe muita capacidade de processamento que não é usada pelos clientes hoje em dia. Isto sugere que o lado do cliente pode assumir mais processamento do que aquele que faz nas arquitecturas cliente/servidor.

- **Tolerância a falhas:** A rede depende de pontos centrais, nomeadamente do servidor, para fornecimento de serviços. Tudo vai abaixo se um servidor central vai abaixo.

- **Expansibilidade:** os sistemas centralizados são difíceis de estender uma vez que os recursos apenas podem ser adicionados ao sistema central.

2.2.2 Arquitectura de rede Peer-to-Peer

As tecnologias para redes distribuídas tentam compensar algumas limitações da arquitectura cliente/servidor. Elas fornecem um cliente com uma interface para chamar serviços mas limitando o conhecimento do cliente sobre onde encontrar os serviços. Os serviços são

distribuídos no meio de servidores e os pedidos dos clientes são enviados para um ou muitos servidores, mas o cliente é incapaz de saber quem lhe está a responder. Mas esta arquitectura classifica os computadores em clientes e servidores, e apenas os servidores podem fornecer uma resposta a um determinado pedido. As tecnologias de redes distribuídas anteriores como o CORBA e DCOM são altamente avançadas e demasiado caras e pesadas para dispositivos integrados. As tecnologias recentes incluíram o ambiente móvel enquanto estavam a ser desenvolvidas, tornando-se, portanto, disponíveis também para os dispositivos móveis.

Uma arquitectura que tem ganho grande relevo nos últimos anos é a arquitectura *peer-to-peer* (P2P). A Figura 2.2 apresenta uma imagem geral desta arquitectura que consiste em nós interligados. O paradigma P2P consiste em tecnologias para redes distribuídas como o JXTA, que será discutido mais adiante neste documento. O *peer-to-peer* surgiu como resultado de diferentes possibilidades:

- **Descentralização:** É um resultado natural das tendências de descentralização na engenharia de *software* com a intersecção da tecnologia existente.

- **Computadores de rede poderosos e largura de banda barata:** Verifica-se o aumento do número de computadores de rede poderosos e da largura de banda barata. O *peer-to-peer* requer efectivamente a disponibilização de *peers* numerosos e interconectados.

- **Programas populares:** Também os aspectos menos técnicos são importantes, como a popularidade de produtos como o Napster, o Gnutella e outros programas similares.

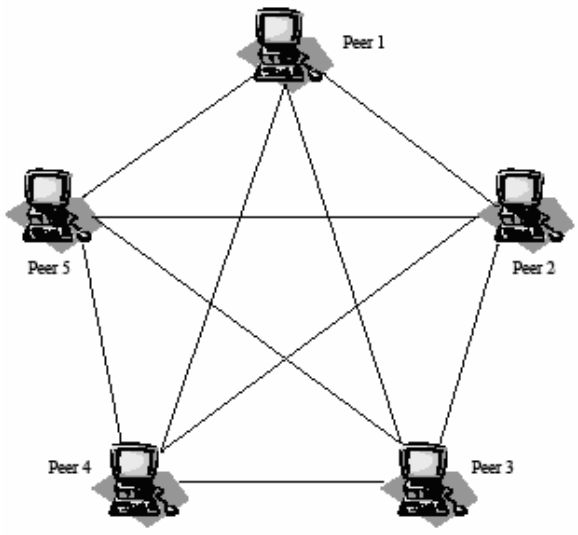


Figura 2-2 Arquitectura *peer-to-peer*: computadores interligados, salientando-se a ausência do servidor

As arquitecturas descentralizadas como o *peer-to-peer* têm praticamente as características opostas dos sistemas centralizados. As propriedades favoráveis da arquitectura *peer-to-peer* são:

- **Capacidade:** Explora a vantagem da capacidade de processamento não usada na *Internet*. Explora a largura de banda disponível na totalidade da rede fazendo uso de uma variedade de canais de comunicação.

- **Independência:** Evita a dependência de um servidor central para fornecimento de serviços.

- **Configuração:** Remove a obrigatoriedade de configurar antes de usar sendo, portanto, classificada como uma arquitectura de rede auto-configurada.

- **Descentralização:** Consiste em nós móveis com *status* idênticos. Cada um pode pedir ou fornecer serviços a outro e criar uma rede sem um controlo central. Reduz o congestionamento da rede uma vez que permite a comunicação via uma variedade de caminhos de rede.

- **Expansibilidade:** Qualquer nó pode juntar-se à rede e instantaneamente tornar disponíveis novos recursos a toda a rede.

- **Tolerância a falhas:** A falha ou o desligar de um nó particular não causa impacto no sistema restante, i.e., um determinado serviço não se torna inacessível pela falha de um simples ponto.

- **Escalabilidade:** Teoricamente, quanto mais "servidores" forem adicionados a um sistema descentralizado, mais capaz este se torna.

As desvantagens de uma arquitectura descentralizada são:

- **Maneabilidade e a coerência na informação:** As redes distribuídas são bastante heterogéneas e, portanto, de difícil gestão. Além disso, os dados do sistema nunca são completamente fiáveis, i.e., não se pode afirmar que um *bit* de dados encontrado é de certeza correcto.

- **Segurança:** As redes tendem a ser inseguras, no sentido em que é fácil a um nó juntar-se à rede e começar a colocar dados suspeitos no sistema.

- **Escalabilidade:** Foi indicada como uma vantagem mas só na teoria. Na prática, os algoritmos requeridos para manter um sistema descentralizado coerente normalmente acarreta bastantes custos. O sistema pode não escalar bem se o custo cresce na mesma proporção do sistema. [Minar2002]

Peer-to-peer é normalmente confundido com rede *ad-hoc*. Uns dizem que o *peer-to-peer* é um paradigma e *ad-hoc* é uma realização do paradigma onde a funcionalidade *peer-to-peer* é colocada nas camadas baixas da pilha do protocolo e, portanto, é dependente de meios físicos. Outro dizem que as realizações *peer-to-peer* podem concentrar-se em fazer a implementação do paradigma independente da tecnologia da rede, resolvendo, assim, os problemas na camada de aplicação. O termo *peer-to-peer* é muitas vezes usado no último caso.

Existem duas formas principais para o paradigma *peer-to-peer*:

- **Puro:** Todos os nós são *peers*, e cada *peer* pode funcionar como um *router*, cliente ou servidor, de acordo com os dados da consulta. Esta foi a forma apresentada anteriormente.

- **Híbrido:** Alguns nós são terminais *routers* que facilitam as interligações entre os *peers*.

Arquitectura *Peer-to-Peer* Híbrida

A noção de *peer-to-peer* tem sido estendida para abranger uma gama de protocolos e soluções que não satisfaz completamente a definição de *peer-to-peer* puro. Muitos protocolos *peer-to-peer* têm introduzido um elemento central na estrutura por forma a oferecer uma conexão consistente. O elemento central pode ser introduzido como uma tabela de *routing* estática ou como um gestor de grupos dinâmico combinado com uma tabela de *routing*. [Skaflestad2001]

Para ser consistente com a ideia de rede *peer-to-peer*, as funções de gestão devem ser atribuídas pelos *peers*. Uma tabela de *routing* central, com mais nenhuma funcionalidade do que o *routing*, é aceitável como um elemento necessário na rede *peer-to-peer* híbrida.

O *router* central pode ser configurado para executar diferentes tarefas. Uma das duas configurações típicas [Graham2001] é como um *router* no sentido tradicional, recebe pacotes, calcula a rota e envia-os aos PCs (ex.: *router* da *Internet*), enquanto a outra solução é mais um serviço de procura (ex.: DNS na *Internet*). A solução que mais se assemelha com *peer-to-peer* é mostrada no modelo 2 da Figura 2.3.

Neste último cenário, o *router* não envia mensagens como um *router* tradicional, mas contém uma tabela de resolução de endereços para assistir os *peers* na procura de outros *peers*. Todas as comunicações são feitas entre os dois *peers* e não através do "servidor" central, e, portanto, o servidor não passa de um mero serviço de procura. Como resultado disto, evitam-se os problemas de congestionamento através de um processamento relativamente pequeno quando se realiza a procura. Uma possibilidade é distribuir esta tabela de resolução de endereços num catálogo distribuído e assim, ser capaz de definir o sistema como um protocolo *peer-to-peer* puro. Problema recorrente irá ser, no entanto: Como adquirir o conhecimento suficiente para registar e comunicar com outro *peer*, e, ao mesmo tempo, garantir uma comunicação fiável?

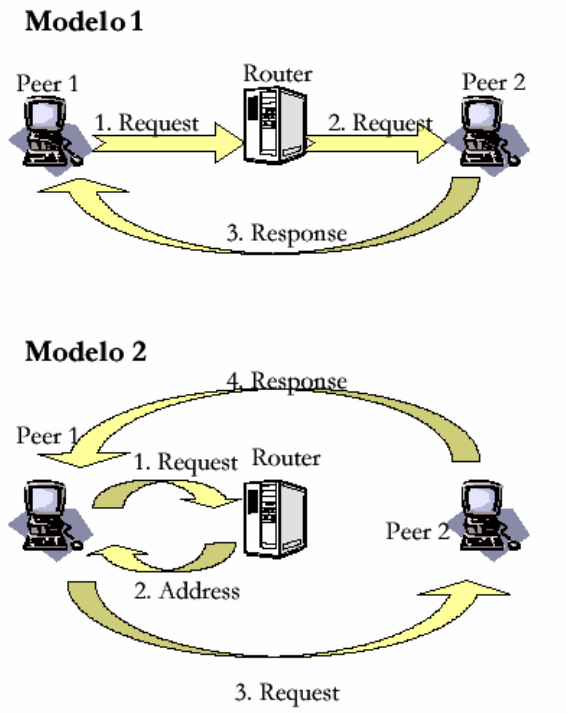


Figura 2-3 *Routers* numa rede *peer-to-peer* híbrida
Modelo 1, o *router* funciona como um *router* da *Internet*
Modelo2, o *router* funciona como um serviço de procura (DNS da *Internet*)

Um sistema que combine um elemento centralizado com um descentralizado adquire algumas vantagens de ambos. Expansibilidade e tolerância a falhas são contribuições da descentralização, enquanto que a centralização contribui com mais coerência que um sistema distribuído puro uma vez que existem poucos servidores que guardam os dados autoritários. Segurança e manuseamento são tão difíceis como nos sistemas descentralizados, mas quando se trata de escalabilidade, já a solução é aceitável. É por tudo isto que esta arquitectura é considerada uma boa arquitectura para *peer-to-peer*. [Minar2002]

2.2.3 Onde Usar a Tecnologia Peer-to-Peer?

A resposta mais óbvia para justificar o uso da tecnologia *peer-to-peer* é a eficiência no uso dos recursos disponíveis da rede. Especialmente, os dispositivos integrados poderão beneficiar consideravelmente usando esta tecnologia, porque podem formar a sua própria rede independentemente das limitações da arquitectura da *Internet*. Nas áreas onde não existem infra-estruturas, as redes *ad-hoc* podem ser usadas como operações de salvamento e,

também, para as pessoas comunicarem de lugares onde existem outros terminais mas não existe infra-estrutura.

Peer-to-peer pode substituir *sites* como o Google, e fornecer procura de informação actualizada. Contrastando com os *sites* de pesquisa que actualizam as suas bases de dados uma vez por dia, ou por semana, cada *peer* será responsável por devolver documentos e outras informações para procura imediata.

Outra área é a computação distribuída. Esta é uma maneira de resolver problemas difíceis dividindo os problemas em subproblemas que podem ser resolvidos de uma forma independente por um grande número de computadores. Até hoje, as aplicações mais populares de computação distribuída não tem sido soluções *peer-to-peer*. Exemplo de um projecto que ganhou muito relevo é SETI@home [SETI] que faz a atribuição de análise de dados do universo e distribuição de dados por computadores recorrendo a *screensavers* por forma a averiguar a existência de outras vidas no nosso sistema solar.

2.3 Aplicações Peer-to-Peer Existentes

Todos estes novos conceitos desencadearam o desenvolvimento de projectos para fornecerem a tecnologia necessária para a sua realização. Nestes incluem-se projectos como o Napster, o ICQ e o Gnutella. Como muitas aplicações de rede *peer-to-peer*, estas não são *peer-to-peer* puras, mas têm uma arquitectura *peer-to-peer* híbrida. Um pequeno resumo destes projectos, que são algumas das aplicações *peer-to-peer* mais conhecidas, será apresentado nesta secção.

2.3.1 Napster

O protocolo Napster [Napster] é composto por clientes e servidores, não parecendo ter, à partida, nada a ver com aplicações de rede *peer-to-peer*. A razão porque o Napster é introduzido como sendo um originador do paradigma *peer-to-peer* é porque se trata do primeiro serviço que faz uso da vantagem da enorme quantidade de espaço de armazenamento livre nos clientes da *Internet*.

Um dos problemas do uso desta capacidade livre foi o facto dos clientes não terem um endereço IP estático como resultado da atribuição pelo ISP de endereços dinâmicos. Isto tornou o DNS impossível para resolução de endereços. A solução encontrada consistiu em

introduzir um protocolo proprietário que contorna o problema do DNS e actualiza o endereço IP dinâmico do cliente em tempo real. Isto foi inicialmente introduzido pelo ICQ em 1996. O protocolo Napster funciona sobre a *Internet* existente e utiliza um *naming service* proprietário, ligando o endereço dinâmico do cliente a um nome específico do Napster. Por forma a realizar esta técnica, o servidor Napster tem que ser incluído em todas as transacções de rede. O servidor Napster oferece um *naming service*, um motor de busca (de ficheiros mp3) e uma aplicação cliente do Napster para uso nos serviços oferecidos pelo servidor.

2.3.2 ICQ

ICQ é uma aplicação onde os utilizadores podem aceder facilmente aos seus contactos, enviar mensagens instantâneas e partilhar ficheiros através da rede. É a primeira aplicação que criou facilmente um endereçamento público de rede uma vez que o ICQ não depende nem de endereços IP, nem de servidores de *domain name*. Apenas dá ao PC um endereço de rede e esse PC pode falar com outro PC com um endereço do ICQ *name space*.

2.3.3 Gnutella

Gnutella [Gnutella] é um programa que oferece partilha, procura e *downloading* de um grande número de tipos de ficheiro. Ao contrário do Napster, o protocolo Gnutella não mantém qualquer forma de *cache* central e não oferece nenhuma nova política de nomes para contornar o problema dos IPs dinâmicos. Uma tabela de endereços IP central, também, não é necessária porque os endereços IP do *peer* são transmitidos quando o *peer* se liga à rede Gnutella. Os *peers* são também ligados a outros *peers* através de uma simples conexão que é iniciada. Mesmo admitindo que o Gnutella não fornece nenhum servidor central, tem o mesmo problema de configuração do Napster: quem contactar?

Por forma a começar uma comunicação, o Gnutella introduz o conceito de pontos *rendezvous*. Um desses pontos tem que ser conhecido pelos *peers* e pode ser encontrado num servidor de um entusiasta Gnutella correndo o gnuCache, que tem algumas semelhanças com a funcionalidade DHCP.

Alguns elementos centrais tiveram que ser adicionados para fazer o protocolo Gnutella correr suavemente, mas além disso, o Gnutella é fortemente baseado na partilha de

recursos armazenados no *peer*. Como tal, o Gnutella tem uma arquitectura *peer-to-peer* híbrida.

2.4 Sumário

Os dispositivos integrados têm recursos limitados e necessitam de tecnologias de rede que possam tratar a mobilidade, i.e., que os dispositivos apareçam e desapareçam da rede sem necessidade de avisar. As aplicações para dispositivos integrados devem ser ligadas a uma rede, por forma a preencher as necessidades dos seus utilizadores: acesso a serviços em qualquer parte, a qualquer hora e em qualquer dispositivo. Java é uma plataforma de desenvolvimento de aplicações que parece ser promissora para dispositivos integrados.

As tecnologias de rede tradicionais como a arquitectura cliente/servidor colocam restrições às aplicações e reduzem a aplicabilidade. Para compensar isto, muitas tecnologias de rede distribuídas foram desenvolvidas, como o CORBA e o DCOM, mas nenhuma delas considera os requisitos dos ambientes móveis antes de ser introduzido o paradigma *peer-to-peer* nas últimas tecnologias de rede distribuídas como o JXTA.

Peer-to-peer é uma tecnologia de rede que não se baseia num servidor central para fornecimento de serviços. Ao invés, os serviços são distribuídos pelos nós participantes e os nós actuam como clientes, *routers* e servidores dependendo do pedido que recebem. As arquitecturas *peer-to-peer* híbridas são dependentes de um servidor central para acesso aos serviços.

Várias aplicações têm sido desenvolvidas que usam a arquitectura *peer-to-peer*, por exemplo o Napster, o ICQ e o Gnutella.

3 Capacidades de ligação em rede do J2ME

Java 2 Micro Edition (J2ME) é a mais recente contribuição da Plataforma Java da Sun que, além desta, é composta pela Java 2 Standard Edition (J2SE) e pela Java 2 Enterprise Edition (J2EE). É uma colecção de APIs (*Application Programimg Interface*) que focam os dispositivos integrados, abrangendo desde os sistemas telemáticos até aos telefones móveis e *Personal Digital Assistants* (PDAs).

Este capítulo faz uma introdução à plataforma Java 2 Micro Edition (J2ME) com foco na *Connected Limited Device Configuration* (CLDC) e no *Mobile Information Device Profile* (MIDP).

A capacidade de conectividade do CLDC e MIDP é a única área que será coberta em detalhe, uma vez que é de interesse para esta tese. Mesmo pensando que as suas capacidades são limitadas, existem soluções e formas de as ultrapassar, como a arquitectura *middleman*. As próximas gerações de MIDP, que fornecerão capacidades e perfis adicionais, estão em desenvolvimento na Java Communiy Process ([JCP], [Lyng2001]).

3.1 Java 2 Micro Edition (J2ME)

A história do J2ME começa no laboratório da Sun onde se pretendia criar uma máquina virtual para o Palm Pilot. Este projecto foi chamado Spotless Project [Spotless] e a máquina virtual criada passou a chamar-se K Virtual Machine (KVM). Esta máquina virtual juntamente com a demo Spotlet foi disponibilizada na JavaOne em 1999. O projecto passou à estandardização via Java Community Process (JCP) e tornou-se num conjunto de configurações e perfis, passando a chamar-se Java 2 Micro Edition.

A Sun fez algumas tentativas no sentido de tornar a máquina de java menor antes da introdução da KVM. Estas tentativas foram o Java Card, o Embedded Java e o Personal Java. J2ME viria a substituir o Embedded java [EJava], enquanto que o Personal Java [PJava] está em vias de estandardização para ser incluído como um perfil no J2ME chamado Personal Profile [J2MEFAQ]. O Java Card [JCard] permite executar programas Java em *smart cards*, i.e., tem objectivos diferentes do J2ME e irá existir como uma tecnologia por si só.

J2ME tem uma pilha de camadas de *software*. São três *layers* no topo do *Host Operation System* do dispositivo. Estes são o *Java Virtual Machine Layer*, o *Configuration Layer* e o *Profile Layer*.

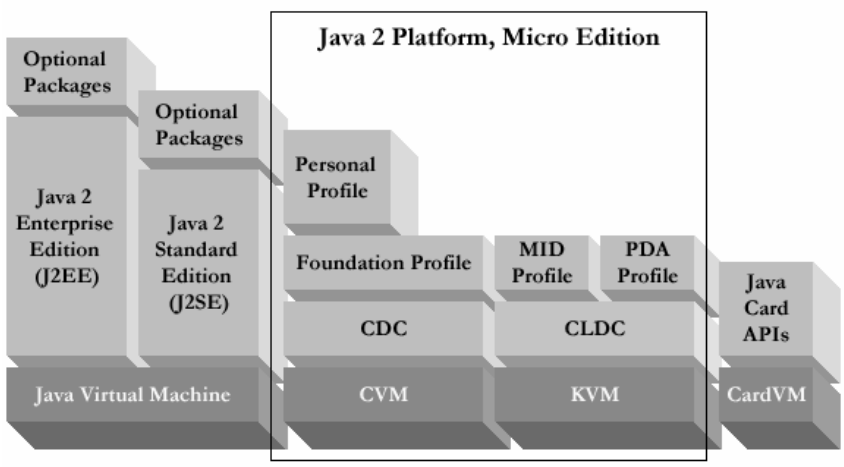


Figura 3-1 Plataforma Java 2, Micro Edition

O *Configuration Layer* engloba as funcionalidades fundamentais no ambiente java e define as funcionalidades mínimas na *java virtual machine* e na *java class library* disponível em todos os dispositivos num vasto espectro de capacidades, também chamado mercado horizontal. Estas funcionalidades são automaticamente incluídas nos profiles que estendem esta configuração particular, e assume-se que estejam presentes para implementação em todos os dispositivos. Hoje, duas configurações estão definidas no J2ME, a *Connected Device Configuration* (CDC) [JSR-36] e a *Connected Limited Device Configuration* (CLDC) [JSR-30]. As duas diferentes configurações são destinadas a duas amplas categorias de produtos:

- CDC foi desenvolvida para dispositivos partilhados, fixos e conectados. Esta categoria de dispositivos tem um vasto número de capacidades de interface com o utilizador, memória entre os 2Mb e os 16Mb e ligações em rede persistentes e com alta largura de banda, na maior parte, usando TCP/IP. Exemplos são as *Set Top Boxes* das televisões, telefones com ecrã baseados na *Internet*, entretenimento para automóveis/sistemas de navegação, etc.

- CLDC destina-se a dispositivos pessoais, móveis e conectados como os telefones celulares, *pagers* e PDAs. Estes dispositivos têm interfaces com o utilizador muito

simples, memória mínima começando nos 128Kb, pouca largura de banda e ligações em rede intermitentes que normalmente não é baseada em TCP/IP.

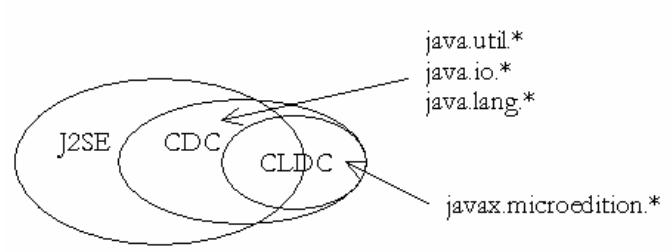


Figura 3-2 Relação entre as configurações J2ME e J2SE: O CDC contém alguns *packages* do J2SE adicionando o javax.microedition. O CLDC é uma versão reduzida do CDC

Uma das chaves para se conseguir ter java a funcionar em dispositivos pequenos é a redução do tamanho das classes de *runtime* instaladas com o ambiente de *runtime*. J2ME foi desenhado do nada, com o objectivo focado na compatibilidade com outras plataformas Java. Comparada a estas, J2ME apenas inclui as classes mais necessárias e representa um subconjunto das classes de *runtime* do J2SE. Por definição, todas as configurações J2ME devem aderir a um encadeamento de relações. Por outras palavras, o CLDC está contido no CDC. Esta relação e a relação com o J2SE são mostradas na Figura 3.2.

O *Profile Layer* define um conjunto mínimo de API disponível numa família particular de dispositivos, i.e., um mercado vertical específico como os telefones móveis e PDAs. É usual que o perfil inclua bibliotecas mais específicas do domínio do que aquelas fornecidas numa configuração. As aplicações são escritas para um perfil particular e implicitamente usa a configuração, uma vez que o perfil está implementado no topo de uma configuração particular. Através deste desenho em camadas, a aplicação torna-se portátil para qualquer dispositivo que suporte o perfil específico para o qual foi escrita. Um dispositivo pode suportar vários perfis.

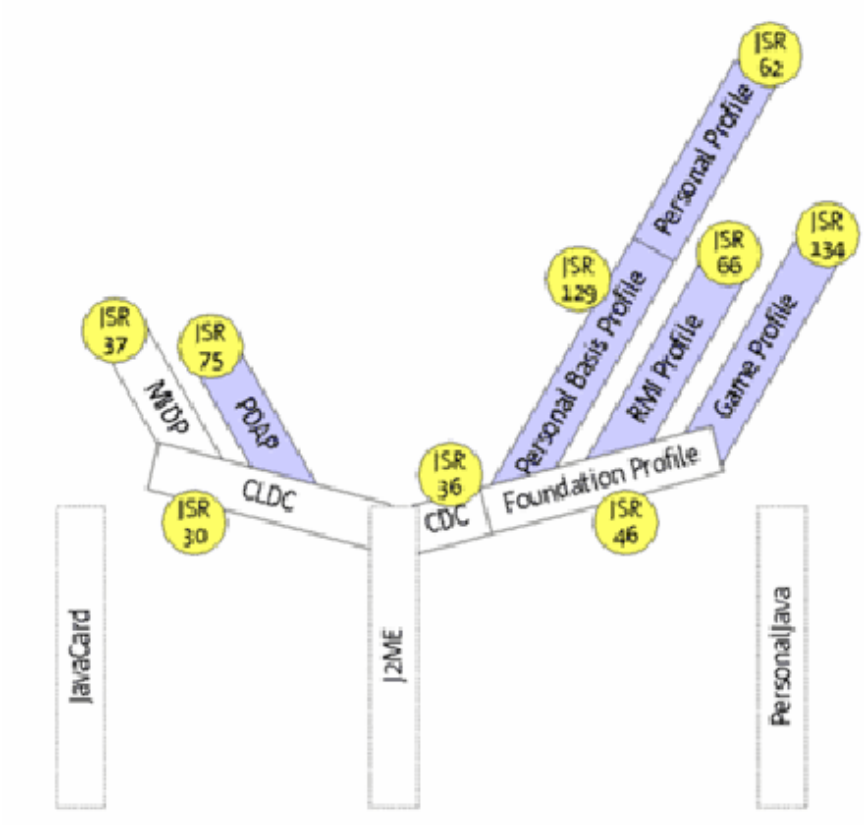


Figura 3-3 A árvore J2ME

Diversos perfis estão em desenvolvimento e as versões 1.0 e 2.0 já estão disponíveis para alguns. A relação entre as diferentes configurações e perfis é mostrada na Figura 3.3, e explicada abaixo.

- Para CDC, três perfis estão, neste momento, a ser estandardizados: o *Personal Basis Profile* [JSR-129] e o *Personal Profile* [JSR-62], o *RMI Profile* [JSR-66] e o *Game Profile* [JSR-134]. Todos eles são baseados na *Foundation Profile* [JSR-46], como é mostrado na figura anterior.

- CLDC tem dois perfis principais e diversos perfis especializados que adicionam funcionalidade aos principais. Os dois principais são o *Mobile Information Device Profile* (MIDP) ([JSR-37], [MIDP1.0], [JSR-118]) , [MIDP2.0], e o *Personal Digital Assistant Profile* (PDAP) [JSR-75].

O foco para esta atribuição são o CLDC e o MIDP e, portanto, o resto do capítulo será dedicado a estas duas especificações. Muitas das tecnologias discutidas em capítulos posteriores estarão também disponíveis para CDC e perfis relacionados.

3.1.1 Connected Limited Device Configuration (CLDC)

A intenção do CLDC é servir como o mínimo denominador comum para diversos tipos de dispositivos com capacidade java e recursos restritos. CLDC necessita ser complementado por perfis uma vez que não é uma solução auto-suficiente e completa. Por exemplo, todos os aspectos relacionados com a interface com o utilizador estão fora do âmbito da especificação CLDC.

Os tópicos principais endereçados pela especificação do CLDC são:

- **Linguagem Java e funcionalidades da máquina virtual**
- **Bibliotecas core do Java** (java.lang.*, java.util.*)
- **Input/Output:** classes para tratamento de *streams* de diferentes tipos
- **Rede:** *framework* genérico para ligações em rede, apresentado na secção 3.2.
- **Segurança:** Discutida na secção 3.6.
- **Internacionalização:** Trata correctamente diferentes codificação de caracteres.

3.1.2 Mobile Information Device Profile (MIDP)

MIDP é um conjunto de APIs do Java que, em conjunto com o CLDC, fornece um ambiente de *runtime* completo para aplicações J2ME destinado aos dispositivos móveis de informação (MID), como os telefones móveis. O objectivo principal do grupo responsável pelo MIDP foi o estabelecimento de um ambiente aberto de desenvolvimento de aplicações de terceiros para MIDs. A memória e os requisitos de rede de um dispositivo MIDP são:

- 128Kb de memória não-volátil
- 8Kb de memória não-volátil para dados persistentes criados pelas aplicações

- 32Kb de memória volátil para o *runtime* do Java, i.e., o Java *heap*.
- O dispositivo deve suportar ligações sem fios *two-way* com largura de banda limitada.

Por forma a atingir portabilidade máxima, a API MIDP considera requisitos absolutos, i.e., não tem requisitos opcionais, como:

- **Aplicações** (i.e. definição da semântica de uma aplicação MIDP e como é que ela é controlada)
- **Interface com o utilizador, ou IU** (inclui a visualização e a interacção)
- **Armazenamento persistente**
- **Rede** (HTTP 1.1)
- ***Timers***

MIPD introduz um novo modelo de aplicação, que omite as restrições da existência de um único ponto de partida. Introduz a máquina de estados que a aplicação pode ter: activa, em pausa e destruída, podendo transitar entre os estados. O objecto central deste modelo de aplicação é fornecer apoio à partilha de dados e recursos, controlada entre múltiplas MIDlets e até em execução simultânea, que é a unidade básica de execução em MIDP.

As MIDlets podem ser empacotadas num arquivo Java (JAR), tanto como uma aplicação simples como um conjunto (*suite*) de MIDlets, para ser executado num dispositivo com capacidades Java. Um ficheiro opcional chamado *Java Application Descriptor* (JAD), que é usado para gerir a aplicação, pode acompanhar o ficheiro JAR. A entrega da aplicação pode ser feita tanto *Over-The-Air* (OTA) como por *downloading* normal para o PC, usando depois um cabo ou conexão IRDA (infravermelhos). OTA não está formalmente incluída na versão 1.0, mas já faz parte da versão 2.0 do MIDP [MIDP2.0].

MIDP 2.0, recentemente lançado, também inclui outros melhoramentos. Segurança no domínio e na rede, funcionalidades de rede adicionais, extensões ao interface com o

utilizador e a inclusão de uma pequena API para som. Também foi sugerido que incluísse um pequeno *parser* XML, mas foi deixado para investigação futura.

A interface com o utilizador inclui extensões de baixo nível para permitir maiores funcionalidades para jogos e controlo da aparência (*layout*) no caso de ecrãs de grandes dimensões. Também foi incluído uma pequena API chamada Mobile Media API [JSR-135] que permite acesso simples e fácil a áudio básico e recursos multimédia. Os itens de segurança adicionados serão discutidos na secção 3.6, enquanto que as extensões à rede serão discutidas no capítulo 3.2.

3.2 Rede utilizando J2ME

As aplicações Java sem fios são, pela sua natureza, centradas na rede. No entanto, os dispositivos em que essas aplicações correm são menos previsíveis. A natureza precisa da ligação da rede depende tanto do dispositivo como dos serviços disponibilizados pela rede na qual estão conectados. Alguns dispositivos podem ser directamente ligados à rede, enquanto que outros apenas são capazes de aceder através de um *gateway*. Independentemente disto, aos dispositivos java sem fios compatíveis com a especificação MIPD é exigido que criem a ilusão de estarem directamente ligados à rede.

As capacidades de rede do J2ME são disponibilizadas pelo *Generic Connection Framework* definido no CLDC. A *framework* dá uma maneira consistente de aceder e organizar os dados num ambiente com recursos limitados. A abordagem usada tem a vantagem de o código da aplicação se manter o mesmo, independentemente do tipo de ligação usada. A implementação do protocolo é deixada aos diferentes perfis.

3.2.1 Introdução

Os requisitos para as bibliotecas de rede e de armazenamento variam conforme os dispositivos e as exigências das diferentes redes para diferentes tipos de comunicação. Os requisitos para se conseguir um pequeno sistema J2ME foram, portanto, deixados à generalização das classes de rede e I/O do J2SE, que é extensível, flexível e coerente no que diz respeito ao suporte de novos dispositivos e protocolos. Em vez de se usar uma colecção de tipos de abstracções totalmente diferentes para diferentes formas de comunicação, é usado, ao nível da programação da aplicação, um conjunto de abstracções relacionadas.

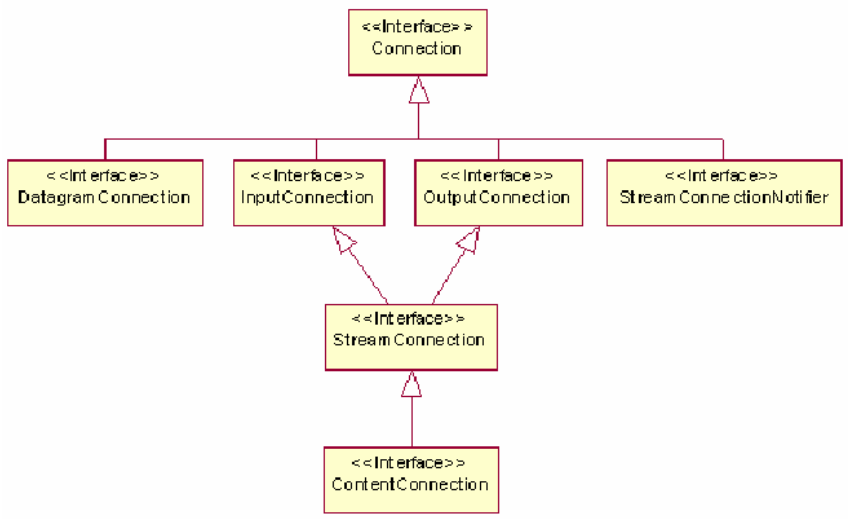


Figura 3-4 CLDC Generic Connection Framework

Este *framework* independente da plataforma fornece as suas funcionalidades sem depender de funcionalidades específicas de um dispositivo. Disponibiliza uma hierarquia de interfaces de conectividades como mostra a Figura 3.4, mas não implementa nenhum deles.

Todas as conexões são criadas usando um único método estático numa classe do sistema chamada Connector e, se houver sucesso, o método devolve um objecto que implementa uma das interfaces de conexão genérica. Este objecto é criado em *background* do *Uniform Resouce Indicator* (URI) fornecido aquando da chamada do método estático. A sintaxe do URI é definida na norma RFC2396 do IETF [RFC2396]. É composta por três partes: um esquema, um endereço e uma lista de parâmetros. A forma geral é:

<esquema>:<endereço>:<parâmetros>

O esquema identifica a forma como a conexão é feita, por exemplo, *socket*, *http*, *file* ou *datagram*, enquanto que o endereço identifica com o que é que se vai conectar e os parâmetros identificam a informação necessária ao protocolo para poder estabelecer a ligação, por exemplo, a velocidade de ligação. Exemplos de URI são http://www.qualquerendereco.com:8080, socket://localhost:8080, datagram://127.0.0.1:8099 [White2002].

Toplay aborda algumas das armadilhas do MIDP no seu artigo [Topley] e aponta três diferenças entre os clientes HTTP escritos em J2SE e os clientes HTTP do MIDP. Indicar porto não é possível uma vez que o suporte HTTP do MIDP actualmente apresenta uma

interface de mais baixo nível do que o seu parceiro J2SE. Isto resulta em que um cliente MIDP terá que tratar detalhes tal como *server redirection*, que é tratado automaticamente nos PCs. É necessário ter em atenção as limitações impostas pelos dispositivos MIDP e adaptar o código.

3.2.2 Protocolos implementados para MIDP

O suporte de rede incluído no MIDP é baseado no *Generic Connection framework* do CLDC. Um requisito para MIDP é o suporte para acesso a servidores e serviços HTTP 1.1. A razão prende-se com o facto de os dispositivos com capacidades MIDP poderem não ter suporte interno para o protocolo IP, e o HTTP poder ser implementado, tanto usando protocolos IP (tal como TCP/IP), como outros (tal como WAP).

Devido à variedade de redes sem fios, pode ser necessária a presença de um *gateway* para fazer a ligação entre o transporte sem fios específico da rede e a *Internet* com fios. Este *gateway* será responsável pela resolução de nomes URL, de tal forma, que o dispositivo possa aceder à *Internet*. Não é obrigatório que a aplicação saiba que tipo de rede está a usar, apesar de poder ter vantagens nesse conhecimento para optimização da transmissão.

A interface HttpConnection fornece a funcionalidade adicional para *request headers*, *parse response headers* e realização de funções específicas HTTP. Cada dispositivo que implemente o MIDP deve suportar conexões abertas usando o esquema URL definido por RFC2616 [RFC2616]. Isto inclui a especificação completa dos pedidos RFC2616 HEAD, GET e POST e as formas absolutas de URLs.

O uso de HTTP torna o dispositivo capaz de chamar serviços da Internet que são baseados no modelo de programação HTTP, tais como *scripts* CGI, PHP e Servlets como é mostrado na Figura 3.5.

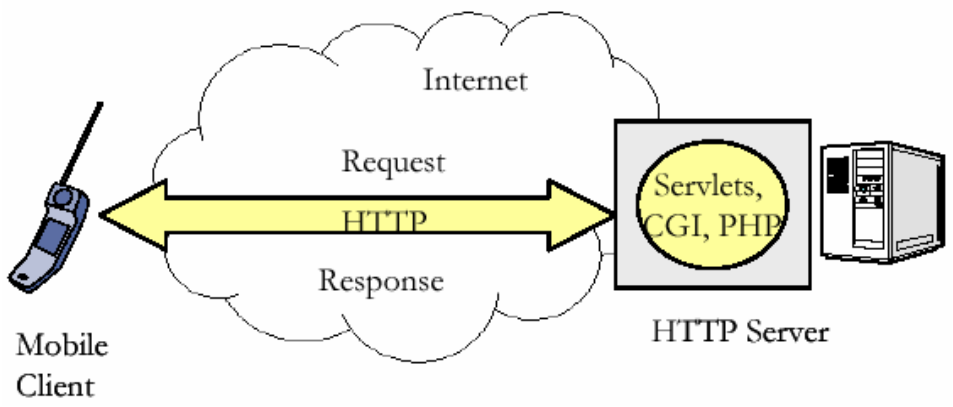


Figura 3-5 Cliente móvel usando HTTP para aceder a serviços na *Internet*

É da responsabilidade daqueles que implementam MIDP o suporte a outros protocolos. O único requisito é que o protocolo deva usar o *Generic Connection framework*. Um exemplo é o Motorola Accompli 008 que disponibiliza suporte a *Sockets* [Acc008].

3.2.3 Suporte aos protocolos recentes

A versão mais recente do MIDP introduziu diversos novos tipos de conexões para dispositivos móveis, mas nem todos são ainda obrigatórios. Alguns irão permanecer opcionais uma vez que o tipo de conexão é independente dos recursos dos dispositivos com capacidades MIDP. E enquanto esses dispositivos tiverem limitações, o HTTP é o único protocolo adequado.

Datagram e *sockets* foram inseridos no MIDP 2.0 e serão discutidos de seguida. Além disso, foram tomadas algumas medidas para maior segurança na rede e o MIDP 2.0 [MIDP2.0] disponibiliza tanto ligações HTTP seguras (HTTPS) como ligações *socket* seguras. Isto será abordado na secção 3.6.

Sockets

Os *sockets* têm a funcionalidade de estabelecer ligações entre dois sistemas e de comunicar como se a ligação fosse um *stream*. Isto é um método primitivo de comunicação mas também leve e útil. Uma vez que os *sockets* têm pouco *overhead*, este pode ser um dos

métodos mais rápidos de troca de dados e emissão de comando entre dois sistemas [White2002].

A contrapartida é que os *sockets* apenas definem a conexão e o mecanismo de transporte de baixo nível, e deixam o cliente e o *listener* do *socket* definirem o protocolo do modo como os dois sistemas irão comunicar. Noutras palavras, os *sockets* dão conexão, mas o formato da troca de informação é deixada ao programador. O resultado é que há poucas restrições no que se pode fazer com *sockets* mas tudo o que se fizer terá que ser determinado e desenvolvido.

Os *sockets* são úteis em casos onde a velocidade é mais importante do que a adesão a um protocolo *standard* aberto, uma vez que usar *sockets* normalmente implica implementar um mecanismo proprietário de transporte de informação. Mas alguns protocolos, como o HTTP, são usualmente implementados usando *sockets*. Se o facto de se usar em standards abertos for um factor importante no desenvolvimento, e o programador não tem comando sobre o servidor e o cliente, os *sockets* serão uma boa maneira de implementar capacidades de comunicação na aplicação. No entanto, existem algumas excepções. Uma vez que os *sockets* apenas fornecem o mecanismo de transporte, outros formatos de dados ou protocolos podem ser conjuntamente usados com os *sockets*, por exemplo XML. O uso do XML faz com que uma aplicação tenha vantagens nos *sockets*, enquanto usar um esquema XML não proprietário. Ainda assim é necessário coordenação entre as aplicações cliente e servidor para assegurar que ambas suportam o mesmo tipo de conexão.

A versão 2.0 do MIDP insere uma classe *socket* e uma classe *server socket*, fazendo com que os dispositivos com capacidades java consigam trocar informação sem o uso de um *middleman* ou *relay*. Os *sockets* podem ser tanto dois telemóveis que trocam mensagens como um dispositivo móvel que se conecta a uma aplicação J2SE ou J2EE num servidor.

As aplicações MIDP que usam *sockets* serão sempre dependentes da infra-estrutura para encontrarem o servidor e vice-versa. Se um dispositivo móvel não se pode conectar a uma rede sem fios, a aplicação não será capaz de trocar mensagens entre o cliente e o servidor.

Datagram

O *datagram* é desenhado para envio de pacotes de dados numa rede. O cliente não é obrigado a estabelecer uma ligação com o servidor antes do envio dos pacotes, i.e., a comunicação é feita sem que exista um canal de comunicação entre o emissor e o receptor. Todas as transmissões do *datagram* são consideradas bem sucedidas. Assim, um *datagram* permite que os dados sejam enviados independentemente de existir ou não um servidor. Portanto, a conexão é considerada pouco fiável, uma vez que os pacotes que se perdem não são reenviados automaticamente pelo protocolo. Não há garantia que os pacotes cheguem pela mesma ordem por que foram enviados, e o protocolo não disponibiliza suporte para "remontar" os pacotes de dados pela ordem correcta. O contrário acontece com os *sockets*, onde a comunicação é orientada pela conexão, significando que a conexão é estabelecida de início e todos os pacotes são "remontados" e retransmitidos sempre que necessário. Se um servidor não suporta *sockets* ou não está à espera de conexões de *sockets*, será gerada uma excepção. Assim, os dados enviados usando *sockets* são considerados fiáveis. Funcionalidades para tratamento da natureza pouco fiável dos *datagrams* podem ser implementadas pela própria aplicação.

Uma das vantagens do uso de *datagrams* é a velocidade. O *datagram* não tem o *overhead* necessário para assegurar que os pacotes cheguem pela ordem correcta ou sequer que cheguem. Para algumas aplicações a velocidade é mais importante do que a integridade dos dados, como o *streaming* de áudio ou vídeo onde a perda de pacotes de dados é normalmente negligenciável.

Existem diversos protocolos para *datagram* mas o mais comum é o *User Datagram Protocol* (UDP). Existem interfaces desenhadas para permitir implementações de diferentes tipos de protocolos *datagram*, como o IP ou o WDP, que juntamente com protocolos proprietários têm a vantagem da natureza do pacote do *datagram* para transmissão de dados. O UDP requer *headers* e metadados mais simples do que o TCP como resultado da sua natureza pouco fiável e é, portanto, mais útil quando a velocidade da entrega é crucial. No ambiente J2ME, o *datagram* pode ser útil devido à sua simplicidade sendo alternativa ao TCP. Por exemplo, pode ser útil para troca de dados entre dois dispositivos através do porto IR.

Outra das funcionalidades do *datagram* é que o programador controla o tamanho do pacote da transmissão. É possível enviar uma grande quantidade de dados num só pacote (até 64Kb). Também é possível enviar um simples *byte* num pacote.

3.3 Cenários de serviços para aplicações MIDP

Depois de descritos os diferentes protocolos para transferência de dados, é tempo de focar as diferentes possibilidades de serviços ou aplicações poderem ser suportados no ambiente J2ME.

Existem diversos cenários possíveis, sendo os quatro principais apresentados nesta secção. Eles diferem nas vantagens e limitações e, alguns podem não ser realizáveis directamente devido ao limitado suporte de rede do MIDP.

3.3.1 Serviço standalone carregado para o dispositivo móvel

A aplicação é disponibilizada por servidores remotos e o terminal móvel estabelece uma ligação e carrega a aplicação desejada, como mostra a Figura 3.6.

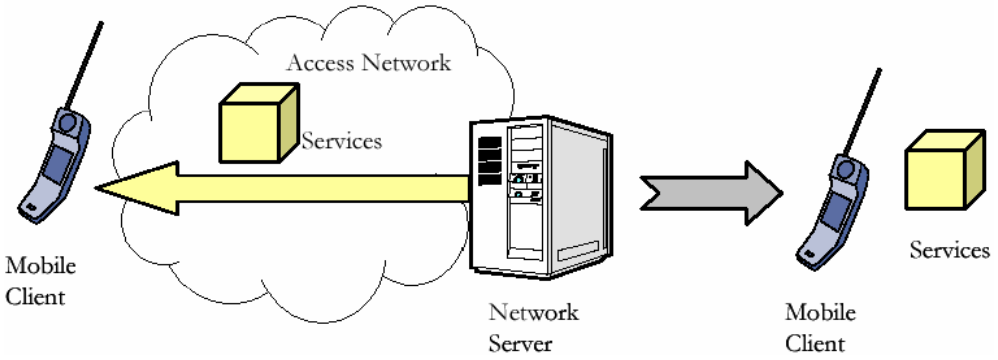


Figura 3-6 Serviço *standalone* carregado para o dispositivo móvel

A aplicação é instalada e configurada no dispositivo móvel e é executada no mesmo, sem ligações à rede. Um jogo é um exemplo de tal aplicação. Devido à lacuna da dependência da ligação à rede, o serviço pode ser executado independentemente de estar ou não a rede disponível.

3.3.2 Serviço executado num servidor remoto

Os serviços são fornecidos e executados no servidor remoto depois do cliente ter estabelecido a ligação. Depois da execução, o resultado é transferido para o cliente e apresentado ao utilizador. O servidor remoto pode estar no domínio do *Network Operator* ou na *Internet*. Aceder a um *website* é um exemplo desde tipo de serviço. Este cenário é visualizado na Figura 3.7.

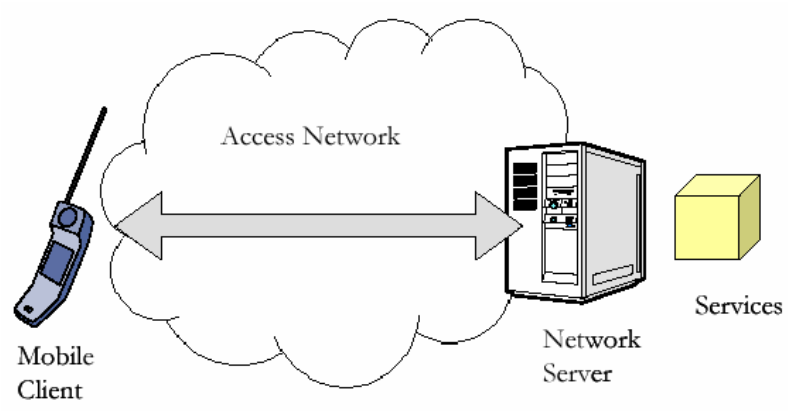


Figura 3-7 Execução de um serviço num servidor remoto

Devido aos recursos limitados do dispositivo móvel, esta é a solução ideal para tarefas computacionalmente pesadas, uma vez que toda a computação é deixada ao servidor e apenas o resultado é devolvido ao dispositivo móvel para posterior tratamento. Assim, apenas uma quantidade reduzida de recursos é usada no dispositivo móvel. A desvantagem é que o dispositivo está dependente do servidor para execução da tarefa e se o dispositivo não tiver ligação à rede este serviço não será executado.

3.3.3 Serviço cliente carregado para o dispositivo móvel

O utilizador carrega uma aplicação que age como um cliente local. Esta aplicação cliente interage com o serviço através de uma conexão com o servidor remoto onde o serviço é fornecido e executado, como mostra a Figura 3.8. Este mecanismo de *download* é disponibilizado pelo OTA da Sun [Lyng2001]. Um cliente de *email* é um exemplo de um serviço.

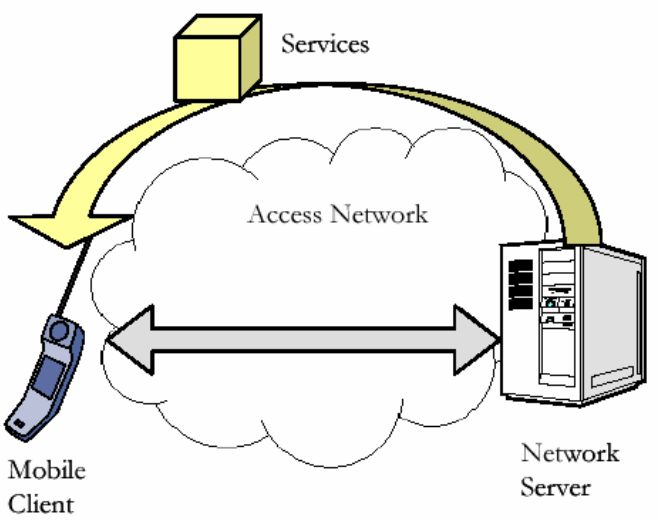


Figura 3-8 Serviço cliente carregado para o dispositivo móvel

A vantagem deste cenário é que a tarefa pesada é deixada ao poderoso servidor mas o próprio cliente pode fazer uma computação menor e, portanto, menos dependente do servidor para completar a tarefa. Também cria possibilidades para o cliente fazer parte da empresa e faz com que o utilizador seja capaz de guardar informação num local e acedê-la independentemente do dispositivo, tempo e local.

Tal como no cenário anterior, este cenário cria uma dependência do servidor central e, como resultado, o serviço é dependente da ligação à rede.

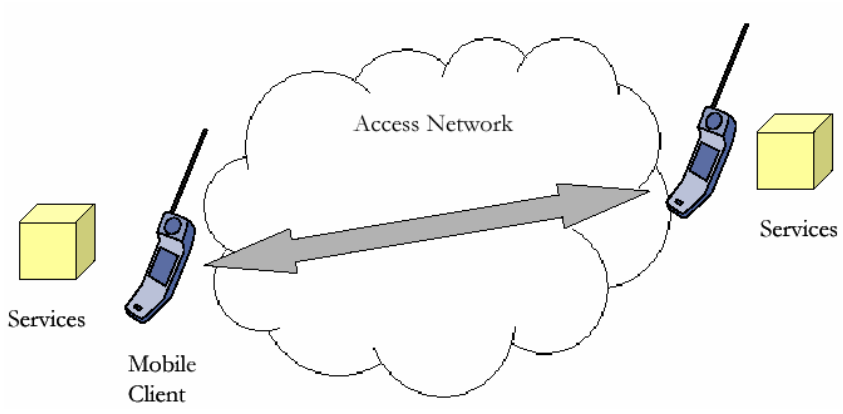


Figura 3-9 Serviços de dispositivo móvel para dispositivo móvel

3.3.4 Dispositivo móvel para o terminal móvel de serviços

Os dispositivos móveis podem requerer o estabelecimento de ligações entre si para disponibilizar, receber e usar serviços interactivos. Os serviços podem ser carregados para os dispositivos através da interacção entre si, ou através de uma combinação de um dos cenários atrás mencionados. Os serviços podem executar localmente sem necessitarem do suporte de servidores. Exemplos de tais serviços são os jogos interactivos e a partilha de informação de agenda.

Uma vez que este tipo de comunicação tem que passar por um ou mais servidores na rede, estes funcionarão como *relays* em benefício do dispositivo móvel. Este tipo de arquitectura, denominada de *middleman*, é descrita na próxima secção. Pretende-se, no futuro, permitir ligações directas entre dispositivos móveis, independentemente da infra-estrutura como a actual rede GSM. Este cenário será abordado em detalhe mais à frente.

3.4 Arquitectura Middleman

Para ultrapassar as limitações da capacidade de rede do CLDC e MIDP, faz-se uso da arquitectura *middleman*. Esta arquitectura é mostrada na Figura 3.10, discutida em [Mahmoud2002], e é usada para permitir às aplicações dos dispositivos móveis usar outras tecnologias de rede além do HTTP.

O CLDC e MIDP não têm suporte para conexões *socket* ou *datagram*. Além disso, a K Virtual Machine (KVM) não suporta todas as funcionalidades da linguagem Java e da Virtual Machine, por ambas implicarem custos de implementação e a sua presença imporia questões de segurança. Como resultado, não há suporte para serialização de objectos e consequentemente não há suporte para *Remote Method Invocation* (RMI).

A Figura 3.10 mostra como uma MIDlet se liga a uma Servlet usando HTTP. A Servlet pode ser ligada a serviços usando *sockets*, CORBA, RMI outras tecnologias do Java. Para aceder a outros serviços, pode-se usar tanto *scripts* CGI como PHP. Desta maneira, a MIDlet será capaz de participar em ambientes distribuídos, fazer parte da rede, e fazer um uso mais amplo como se o terminal acedesse aos serviços directamente através de HTTP.

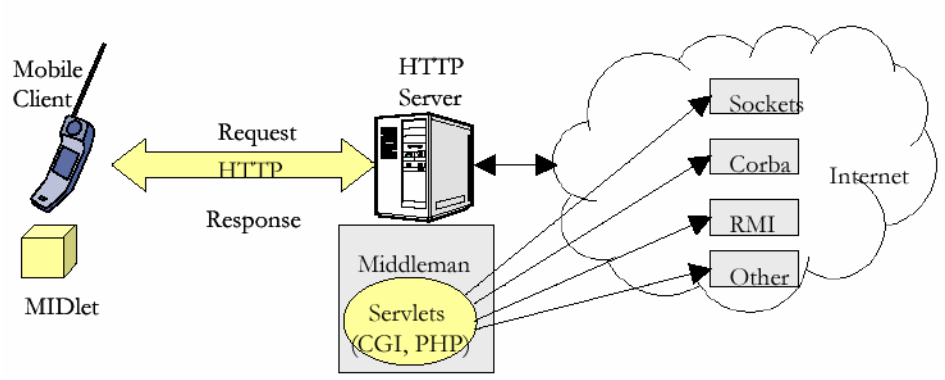


Figura 3-10 Arquitectura *Middleman*

As vantagens da arquitectura *middleman* são

- Não dependência de IP, uma vez que nem todos os dispositivos o implementam
- Promove acoplamento solto simplificando as interacções entre objectos
- Permite ao utilizador usar RMI, CORBA ou outra tecnologia distribuída a partir do dispositivo MIDP.

Em contrapartida a MIDlet tem que conhecer o endereço do *relay*, ou o utilizador tem que fornecer o endereço à aplicação. Ambas as soluções constituem uma implementação pouco flexível.

3.5 Novas tecnologias de rede para dispositivos J2ME

Diversas tecnologias de rede estão em desenvolvimento e serão disponibilizadas, no futuro, nos dispositivos integrados, móveis e com capacidades J2ME. Algumas estão em desenvolvimento no *Java Community Process* (JCP) [JCP], enquanto que comunidades independentes desenvolvem outras.

Pacotes da JCP incluem o uso de *Short Message Service* (SMS) e tecnologias suplementares. Outros incluem uma Location API que pode ser útil e uma SIP API, sendo todas de menor importância. Diversos pacotes e perfis adicionais estão em desenvolvimento para J2ME e alguns estão quase concluídos. Uma revisão dos diversos *Java Specification Requests* (JSRs) relacionados com J2ME podem ser encontrados em [J2ME_spec]. Uma API para *Bluetooth*

foi lançada na Primavera de 2002 e a última parte deste sub-capítulo fará uma breve introdução às funcionalidades principais.

A arquitectura de rede JXTA é desenvolvida por uma comunidade, a comunidade JXTA. Introduz algo de novo à rede e é parte do novo paradigma chamado tecnologias de rede *peer-to-peer*. Foi originalmente desenvolvido para dispositivos que corressem J2SE ou J2EE mas, mais tarde, foi suplementado com versões compactadas para serem usadas com J2ME. Enquanto que o *Bluetooth* abrange *peer-to-peer* em todas as camadas da pilha de protocolos, o JXTA tem como objecto apenas a camada da aplicação.

Espera-se que a próxima geração de telefones móveis seja independente da tecnologia da transmissão, i.e., que os próprios telefones descubram a tecnologia mais apropriada à sua posição actual e, quando se começarem a afastar, ou quando o sinal de outra tecnologia ficar mais forte, devem trocar directamente sem qualquer interacção por parte do utilizador.

3.5.1 JXTA

O projecto JXTA define um conjunto de protocolos que torna possível operar numa base *peer-to-peer* e um formato de mensagem para ser usado na comunicação. Como resultado, JXTA é independente da linguagem e da plataforma de implementação. A referida implementação é em Java mas existem outros projectos para implementação dos protocolos em C e noutras linguagens. Uma apresentação mais aprofundada do JXTA será feita e discutida tanto genericamente como para dispositivos J2ME no capítulo 4.

3.5.2 Bluetooth

Bluetooth é uma especificação *point-to-point* de rede sem fios, destinada a substituir as conexões portáveis por cabo e/ou dispositivos electrónicos fixos. As funcionalidades principais são a robustez, baixa complexidade, pouca necessidade de energia, baixo custo e alta velocidade. É destinada a ser usada como uma *Personal Area Network* (PAN), ligando dispositivos como telemóveis, PDAs, impressoras e *headsets* quando estão próximos uns dos outros.

O *standard Bluetooth* está dividido em dois grupos: *core* e *profile*. As especificações *core* descrevem os detalhes das várias camadas do protocolo *Bluetooth*, da interface rádio ao

controlo da ligação. Tópicos relacionados são também cobertos, tais como, a interoperabilidade com as tecnologias relacionadas, requisitos de teste e uma definição de vários *timers Bluetooth* e os seus valores associados.

O segundo grupo, a especificação *profile*, está incumbida do uso da tecnologia *Bluetooth* para suporte a várias aplicações. Cada especificação *profile* discute o uso da tecnologia definida na especificação *core* para implementar um modelo de uso particular e incluir uma descrição de quais os aspectos das especificações *core* que são obrigatórios, opcionais ou não aplicáveis. O propósito de uma especificação *profile* é descrever uma interoperabilidade *standard* por forma a que os produtos de diferentes vendedores funcionem em conjunto. Em geral, as especificações *profile* caem numa de duas categorias: substituição do cabo ou rádio sem fios.

As APIs do Java para *Bluetooth Wireless Tecnology* [JSR-82] estão desenvolvidas para serem usadas no CLDC e cobrem transmissões de dados puras. A intenção desta API é suportar o registo de serviços e a descoberta de dispositivos e serviços. É capaz de estabelecer ligações RFCOMM, L2CAP e OBEX e conduzir todas estas actividades de forma segura.

Os protocolos abrangidos são:

- **L2CAP** (Logical Link Control and Adaptation Protocol)
- **RFCOMM**
- **SDP**
- **OBEX**

Os perfis incluídos na implementação J2ME do *Bluetooth* estão discriminados abaixo:

- **Generic Access Profile** (GAP)
- **Service Discovery Application Profile** (SDAP)
- **Serial Port Profile** (SPP)
- **Generic Object Exchange Profile** (GOEP)

3.6 Segurança

A segurança em MIDP é limitada sendo as MIDlets uma potencial ameaça para os utilizadores. Mas existem ainda outros riscos, não tanto no dispositivo móvel, mas na comunicação com entidades na rede.

Mesmo pensando que as MIDlets têm grandes oportunidades quando se juntam à rede, um grande número de aplicações não serão viáveis sem alguma forma de segurança de dados. Qualquer informação transmitida através de uma ligação sem fios está sujeita a ser interceptada. Alguma dessa informação pode ser sensível, como o número de cartões de crédito ou outras informações pessoais. A solução necessária depende do nível de sensibilidade. Por forma a disponibilizar uma solução completa de segurança entre dois pontos, é necessário implementá-la em ambos os pontos, cliente e servidor, e assegurar que os sistemas intermediários também são seguros. Uma solução a ter em conta quando se lida com informação muito sensível é a encriptação: o emissor encripta os dados antes de os transmitir na ligação sem fios; o receptor autorizado recebe os dados encriptados e descodifica-os usando uma chave apropriada.

Esta secção considera as limitações do MIDP. Depois a atenção recairá sobre a comunicação em rede e a criptografia e quais os pacotes de segurança disponíveis no MIDP, ou em versões posteriores.

3.6.1 Aplicação de segurança no dispositivo

A segurança em J2SE, apesar de poderosa e flexível, excede a quantidade de memória disponível nas máquinas virtuais que suportam CLDC. Também requer alguma administração que pode estar além do conhecimento esperado de um utilizador de dispositivos móveis.

O CLDC e o MIDP trazem segurança de baixo nível para KVM e segurança ao nível da aplicação. A segurança de baixo nível é conseguida através da verificação das classes java, enquanto que a segurança ao nível da aplicação prende-se com a execução da aplicação nos dispositivos com capacidades java.

De forma a assegurar que a KVM não provoca nenhum dano ao dispositivo em que corre, as classes java devem ser verificadas. Esta verificação assegura que os ficheiros Java *.class*

não podem conter referências a locais de memória inválidos. No J2SE esta verificação é feita antes da execução mas nos dispositivos MIDP com limitações de memória, as classes devem ser pré-verificadas, significando que o processo de verificação ocorre antes do *downloading* da aplicação. Antes da execução, o processo de verificação interno ao dispositivo é feito usando a informação gerada pela ferramenta de pré-verificação.

Quando o CLDC foi desenhado, o modelo de segurança foi mantido simples, sendo esperadas soluções de segurança mais eficazes em versões posteriores da especificação CLDC [Lyng2001]. A KVM inclui uma *sandbox* simples, onde as aplicações java devem ser executadas assegurando um ambiente fechado, no qual a aplicação só pode aceder às APIs que foram definidas pela configuração, perfis e classes de licença aberta suportadas pelo dispositivo. Mas uma confirmação segura das chamadas à API, existentes no J2SE, não estão incluídas, salvo raras excepções. Como o MIDP não suporta nenhuma funcionalidade adicional relativamente às disponibilizadas pelo CLDC, faz com que praticamente não exista nenhuma segurança que possa ser tratada pelo utilizador. [Topley2002]. Assim, as MIDlets são uma potencial ameaça ao utilizador.

A segurança limitada incluída assegura que as MIDlets não tenham acesso a informação armazenada no telefone, como por exemplo, as listas telefónicas ou agendas e não estejam autorizadas a ter controlo directo sobre o dispositivo. Os dados guardados por uma MIDlet são privados dessa MIDlet e da sua *suite*. Assim, apenas pode danificar os seus próprios dados.

3.6.2 Segurança na rede e soluções de encriptação

Quando se transmitem dados através de uma rede é possível que os mesmos sejam interceptados a partir de vários pontos da rede. Esta é a razão para a existência de uma solução que mantenha os dados sensíveis fora do alcance de potenciais intrusos. Uma solução para este problema é o uso de criptografia.

A criptografia é a ciência da escrita secreta e é um ramo da matemática que é baseada na ideia de que certos tipos de problemas matemáticos são muito difíceis de resolver. À medida que a investigação matemática continua, é possível que alguém possa vir a descobrir uma maneira de resolver a maior parte dos algoritmos modernos de criptografia.

Mas, até à data, eles fornecem protecção para os dados sensíveis e não há, também, grandes alternativas. ([Knudsen2001], [Gollmann1999]).

A espionagem em ligações inseguras pode ser combatida por serviços e mecanismos de segurança da comunicação. Os serviços ou aspectos importantes de segurança incluem:

- **Confidencialidade**: outras pessoas não devem ser capazes de aceder a informação sensível que é enviada na rede
- **Integridade**: assegurar que os dados não são alterados ou corrompidos
- **Autenticação**: o processo de fornecer identidade.

A criptografia fornece soluções a cada um desses aspectos de segurança: *Integrity Check Functions*, assinaturas digitais e algoritmos de encriptação (*ciphers*) ([Knudsen2001], [Gollmann1999]).

- **Integrity Check Functions:** aplica uma função de *hash* a uma grande quantidade de dados, de forma a transformá-la numa quantidade menor de dados. Se apenas um bit dos dados originais for alterado, resultará num valor totalmente diferente. O resultado da aplicação de uma função de *hash* tem diversos nomes, como valor de *hash*, *message digest* e *checksum*. Às vezes, é também referido por *digital fingerprint*. Um exemplo de uma função de *hash* é o *Secure Hash Algorithm* (SHA-1).
- **Assinaturas digitais:** uma assinatura digital é como uma função de verificação da integridade excepto no facto de ser produzida por uma pessoa particular. O esquema consiste num algoritmo de assinatura e num algoritmo de verificação. Para criar a assinatura, é necessário que a pessoa tenha uma chave privada, enquanto que uma chave pública correspondente pode ser usada por qualquer pessoa para verificar se a assinatura veio da pessoa certa. Os certificados são uma extensão das assinaturas digitais e traduzem-se em documentos assinados por uma autoridade que prova a sua identidade. Alguns exemplos são El Gamal Signatures, Digital Signature Algorithm (DSA) e o algoritmo desenvolvido por Ronald **R**ivest, Adi **S**hamir, and Leonard **A**dleman (RSA).

- **Algoritmos de encriptação (*ciphers*):** Um algoritmo de encriptação pode ser usado para encriptar e decriptar dados. Encriptação significa a transformação dos dados iniciais em texto ilegível, chamado *cipher text*. Este *cipher text* pode ser decriptado de forma a obter-se novamente o texto inicial. Para isso são precisas chaves. O uso de chaves no *cipher text* será diferente para cada chave. Nos algoritmos simétricos, é usada a mesma chave para encriptar e decriptar, enquanto que nos algoritmos assimétricos (ou algoritmos de chave pública), são usadas chaves diferentes para encriptar e decriptar. Os *Ciphers* podem operar de diferentes modos que determinam como é que o texto irá ser encriptado e transformado em *cipher text*. Exemplos são *Data Encryption Standard* (DES) que é simétrico e a encriptação RSA que é assimétrico.

Só se deve efectuar a instalação de MIDlets quando o *software* vem de fontes de confiança mas, hoje em dia, não existem mecanismos que assegurem ao utilizador quem fornece a MIDlet. O J2SE inclui um mecanismo de autenticação, i.e., criptografia de chave pública e certificados, mas esta solução não vem incluída na especificação MIDP 1.0. O MIDP 2.0 tem um modelo de segurança melhorado que inclui a assinatura de aplicações e verificação de certificados. Desta forma, o utilizador será capaz de verificar qual o fornecedor da MIDlet.

O suporte à criptografia está disponível no J2SE através da *Java Cryptography Architecture* (JCA) e *Java Cryptography Extension* (JCE). Mas ambas são muito pesadas para a plataforma MIDP. Até agora não existe qualquer solução para criptografia no J2ME da Sun. Existem outras soluções como o pacote de criptografia da Bouncy Castle [Bouncy], um esforço *open source* desenvolvido na Austrália. Alguns consideram que é um excelente trabalho de criptografia, caracterizado por um API limpo e uma formidável *toolbox* de algoritmos de criptografia [Knudsen2001]. A Bouncy Castle oferece uma distribuição muito leve do seu *software* para J2ME.

Existem soluções de segurança para alguns dos problemas encontrados no MIDP do J2ME que não envolvem o tratamento da segurança ao nível da aplicação. Ainda continua a ser encriptação mas é tratada como parte da pilha de protocolos responsáveis pelo envio de dados na rede antes desses dados serem enviados. Aplicações de comércio electrónico (*e-commerce*) usam o *Secure Socket Layer* (SSL) desenvolvido pela Netscape e a esperança é que

funcione também para *m-commerce* (*mobile commerce*) [Mahmoud]. A Sun tem estado a trabalhar numa versão *light* do SSL chamada kSSL e está incluída na especificação MIDP 2.0 que é capaz de suportar a versão segura do protocolo HTTP, HTTPS, e conexões *sockets* seguras ([Mahmoud2000], [Gupta2001]).

Por forma a melhorar ainda mais a segurança no J2ME, uma API está em desenvolvimento pela *Java Community Process* (JCP) [JCP], JSR 177 *Security* e *Trust Services* API para J2ME [JSR-177]. É proposto como um pacote opcional para ser usado em conjunto com diversos perfis J2ME. O objectivo é definir uma colecção de APIs que forneça serviços seguros para dispositivos J2ME. Isto tornará os dispositivos mais seguros e com mecanismos para suporte a uma ampla variedade de aplicações baseadas em serviços como o acesso à rede empresarial, *mobile commerce* e *digital rights management*. Muitos serviços baseiam-se na interacção com um elemento de segurança no dispositivo para armazenamento seguro e execução segura. Pode ser armazenamento seguro para protecção de dados delicados como chaves privadas do utilizador, certificados de chave pública, credenciais de serviços e informação pessoal. Outro serviço pode ser o assegurar uma execução segura como operações criptográficas para suporte a protocolos de pagamento, integridade de dados e confidencialidade de dados. E por último, adicionar funcionalidades de segurança em que as aplicações J2ME possam confiar para tratamento de serviços, tais como identificação, autenticação, *banking*, pagamentos, compra de bilhetes e visualização de *medias* digitais. É sugerida a implementação de elementos de segurança em *smart card*, à medida que são amplamente usados em telefones sem fios, como os cartões SIM nos telefones GSM. Assim, a especificação prevê um modelo de acesso que faça com que aplicações de dispositivos J2ME comuniquem com o *smart card* inserido nesse dispositivo [JSR-177].

3.7 Sumário

J2ME é uma colecção de APIs que foca os dispositivos integrados e de consumo. As componentes principais são a configuração e perfis (*profiles*), onde a configuração cobre as funcionalidades fundamentais no ambiente java e os perfis definem funcionalidades adicionais como a interface com o utilizador, armazenamento e capacidades de rede. Uma das duas configurações é a *Connected Limited Device Configuration* (CLDC), que é o fundamento para o *Mobile Information Device Profile* (MIDP), o perfil focado no resto do capítulo.

As capacidades de rede do MIDP são construídas sobre o *Generic Connetion Framework* definido no CLDC que fornece uma maneira consistente de aceder e organizar dados num ambiente com recursos limitados. O MIDP implementa o protocolo HTTP 1.1, embora mais suportes de *sockets* e *datagrams* estejam disponíveis na mais recente versão do MIDP. Serviços ou aplicações, podem ser usados e acedidos de diversas maneiras usando MIDP. Ambos através do *download* da aplicação para o dispositivo para execução local, para serem executados num servidor remoto e apenas devolverem os dados processados ao dispositivo móvel, ou uma combinação destas duas abordagens. A esperança é a de que, no futuro, dois telemóveis sejam capazes de comunicar independentemente da infra-estrutura. Hoje em dia, as MIDlets são dependentes de um *relay* que actua em seu benefício para acesso à rede. Mas, assim, permitem que as MIDlets usem tecnologias de rede mais avançadas como o RMI e CORBA. Novas tecnologias que usam esta arquitectura são o Jini e o JXTA. O uso de *Bluetooth* remove esta dependência de um *relay* mas limitará o dispositivo na medida em que para todas as ligações à rede apenas use *Bluetooth*.

O MIDP também está limitado no que diz respeito à segurança, especialmente nas ligações a redes sem fios. Assim, são necessários pacotes para assegurar integridade, confidencialidade e autenticação. Mas não existem. Uma API que está em desenvolvimento, mas, entretanto, um projecto *open source*, o Bouncy Castle, fornece um pacote criptográfico para J2ME. Mas a adição de criptografia a uma aplicação ou sistema não o torna, necessariamente, mais seguro. É necessário, também, efectuar uma abordagem de segurança ao nível do sistema e a criptografia é apenas uma das ferramentas da caixa. O MIDP 2.0 melhorou a segurança com a introdução de certificados e verificação de MIDlets e SSL para tornar as conexões *http* e *sockets* mais seguras.

4 JXTA

JXTA, abreviatura de *Juxtapose*, é um projecto *open-source* que define um conjunto de protocolos para computação *peer-to-peer ad-hoc*. Começou como um projecto de investigação chamado *Juxtapose* na Sun Microsystem [Sun] no Verão de 2000, chefiado por Bill Joy. Em Abril de 2001, o projecto continuou a ser desenvolvido numa comunidade *open-source*, a comunidade JXTA, onde qualquer um podia contribuir para o desenvolvimento.

JXTA esforça-se por fornecer uma estrutura base P2P sobre a qual podem ser construídas outras aplicações P2P, visto que os *standards* estão ausentes no mundo P2P. Soluções P2P correntes usam protocolos, arquitecturas e implementações diferentes. A base do JXTA consiste num conjunto de protocolos que são independentes da linguagem, independentes da plataforma e agnósticos da rede, i.e., eles não assumem nada sobre a rede subjacente. Cada protocolo serve um propósito especial e é usado para troca de mensagens entre as entidades na rede P2P.

Este capítulo dará uma breve introdução ao JXTA, aos seus objectivos, suas terminologias e conceitos, a arquitectura e protocolos. Um dos projectos da comunidade JXTA é o desenvolvimento de uma implementação do JXTA para dispositivos que suportam J2ME. Uma descrição desta implementação também será apresentada.

4.1 Introdução ao JXTA

O projecto JXTA prevê um mundo onde cada *peer*, independentemente do *software* e da plataforma *hardware*, pode beneficiar e lucrar por estar conectado com milhões de outros *peers*. O JXTA, como consta na declaração sobre a visão, está caracterizada abaixo.

"O projecto JXTA está a construir uma tecnologia computacional de rede para fornecer um conjunto de mecanismos simples, pequeno e flexível que pode suportar computação P2P em qualquer plataforma, em qualquer lado e em qualquer lugar. O projecto está primeiramente a generalizar as funcionalidades P2P e a construir um núcleo de tecnologias que considera as limitações da computação P2P. O foco está na criação de mecanismos básicos e deixando as escolhas de políticas aos criadores das aplicações." [Krishnan2001]

Os objectivos do Projecto JXTA são baseados naquilo que são consideradas deficiências em muitos sistemas *peer-to-peer* ([Comp2001],[Gong]):

- Interoperabilidade
- Independência da plataforma
- Ubiquidade

Interoperabilidade no JXTA é conseguida desenhando a estrutura por forma a permitir que *peers* interconectados localizem facilmente outro *peer*, comuniquem um com o outro, participem em actividades baseadas numa comunidade e ofereçam serviços uns aos outros de uma forma semelhante em sistemas P2P diferentes e em comunidades diferentes. Muitos sistemas *peer-to-peer* são construídos para entregar um determinado tipo de serviço. O Napster, por exemplo, fornece a partilha de ficheiros de música, ICQ fornece mensagens instantâneas e o Gnutella fornece a partilha de ficheiros genéricos. Estes sistemas são incompatíveis devido à ausência de uma infra-estrutura P2P subjacente comum. Cada fornecedor cria a sua própria comunidade de utilizadores P2P, duplicando o esforço para a construção de *software* e primitivas de sistema comummente usadas por todos os sistemas P2P. Os programadores que pretendam oferecer o mesmo serviço em duas comunidades diferentes terão que desenvolver o mesmo serviço duas vezes, ou desenvolver um sistema de ponte entre as comunidades.

A tecnologia JXTA é desenhada para ser independente da plataforma, i.e., é independente da linguagem de programação, plataformas de sistema e plataformas de rede. A API de muitos sistemas P2P é baseada num sistema operativo particular e num protocolo de rede particular, e como resultado, é improvável que dois sistemas interoperem.

Além disso, o JXTA oferece ubiquidade. É desenhado para ser implementado em qualquer dispositivo digital, incluindo sensores, electrodomésticos, PDAs, aparelhos, *routers* de rede, computadores, servidores de dados e sistemas de armazenamento. Os projectistas de P2P usualmente procuram obter lucro o mais depressa possível, tendo como alvo o maior número de consumidores e fazendo a escolha do Microsoft Windows como a plataforma alvo. Como resultado muitas dependências das funcionalidades do específico Wintel (Windows-Intel) são transportadas para o sistema. Como é muito provável que outros

dispositivos e sistemas venham a ter grande proveito da tecnologia P2P, a aposta num segmento particular de *hardware* e sistema de *software* não será muito provável.

O JXTA tem muitas vantagens e limitações, algumas foram encontradas enquanto estudava o JXTA, outras são apontadas por [Wilson2002]. As vantagens são:

- Esforça-se por fornecer uma maneira *standard* de comunicar numa rede P2P
- Fornece uma linguagem mais abstracta para comunicação entre *peers* do que os anteriores protocolos P2P especializados, permitindo uma maior variedade de serviços, dispositivos e transportes de rede.
- O JXTA não determina uma linguagem de programação ou um ambiente em particular.
- *Extensible Markup Language* (XML) ou representação binária de dados é usada para trocas de mensagens, ambas compreendidas pela maioria das plataformas disponíveis actualmente. O XML fornece um formato *standard* para estruturas de dados que são bem conhecidas, bem suportadas e facilmente adaptadas num formato humanamente legível, tornando fácil aos programadores o *debug* e compreensão.
- Fornece funcionalidade P2P a "todos os dispositivos digitais"
- O JXTA está disponível a *peers* por trás de *firewalls* e NATs.

Limitações do JXTA:

- Muitos queixam-se que a iniciativa do JXTA na criação de um *standard* P2P aparece demasiado cedo, uma vez que o *peer-to-peer* ainda não teve tempo de amadurecer.
- O JXTA constitui uma estrutura de trabalho extensiva. Isto pode prevenir o uso da especificação uma vez que há muito que aprender antes de ser usado.
- O JXTA não tenta indicar como é que os serviços são invocados com excepção dos serviços centrais. Existem muito *standards* para invocação de serviços, como a *Web Service Description Language* (WSDL), mas nenhum foi escolhido especialmente pelo

grupo de especificação de protocolo do JXTA. O JXTA disponibiliza uma estrutura genérica para trocas de informação entre os *peers*. Assim, qualquer mecanismo, como o WSDL, pode potencialmente ser usado via JXTA para troca da informação requerida para invocar serviços.

- Os projectistas do JXTA incluíram várias flexibilidades através da especificação do protocolo JXTA e alguns criticaram esta flexibilidade. Embora o uso do XML pelo JXTA especifique todos os aspectos da comunicação P2P para qualquer aplicação genérica P2P, o JXTA pode não ser adequado para uma aplicação P2P *standalone* específica. Numa aplicação individual o custo de rede pelo uso do XML pode ser algo problemático, especialmente se o programador não tem a intenção de tirar partido das vantagens das capacidades do JXTA em incorporar outros serviços P2P na sua aplicação.

- A abstracção da plataforma do transporte de rede tem sido criticada como uma área potencial de excesso. Se muitas aplicações P2P actuais dependem do protocolo IP para fornecer o transporte na rede, porque é que o JXTA vai tão longe no evitar amarrar os protocolos num transporte de rede específico? Porque é que não especifica o IP como um transporte de rede assumido, eliminando os custos?

- Os protocolos JXTA não prometem por si só a interoperabilidade. Só porque duas aplicações são construídas sobre o JXTA não quer dizer que elas irão magicamente interoperar. Os programadores devem desenhar aplicações para serem interoperáveis. No entanto, podem usar o JXTA, que disponibiliza a camada base interoperável, para aumentar a interoperabilidade futura.

Os programadores devem equilibrar a flexibilidade com a performance quando implementam as suas aplicações P2P. O JXTA fornece a plataforma mais conseguida para a produção de aplicações P2P que têm a flexibilidade requerida para crescer no futuro. Mas isto não indica que é a melhor ou a solução mais eficiente para implementar aplicações P2P particulares. O valor central da plataforma JXTA é a capacidade de influenciar outros serviços P2P e permitir o desenvolvimento de comunidades P2P. Enquanto a tecnologia estiver no seu estágio inicial, é esperado que amadureça com o tempo para permitir uma estrutura robusta e fiável para computação P2P.

A comunidade JXTA tem discutido a possibilidade de submeter os protocolos JXTA a uma organização externa e passar à estandardização formal.

O resto desde capítulo irá focar a especificação do JXTA e a implementação da especificação, que é escrita em Java e impõe algumas dependências. Por exemplo, os protocolos JXTA são implementados para suportar o transporte HTTP e TCP/IP.

4.2 Arquitectura

A arquitectura JXTA consiste em três camadas: a camada plataforma, a camada serviço e a camada aplicação como se pode observar na Figura 4.1.

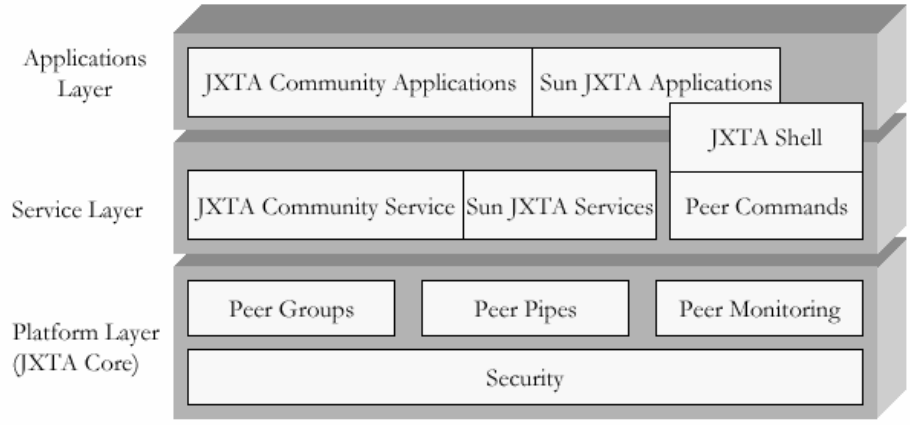


Figura 4-1 A arquitectura de 3 camadas do JXTA

A fronteira entre serviços e aplicações não é rígida. Uma aplicação para um determinado utilizador pode ser vista por outro como um serviço. Todo o sistema é desenhado para ser modular permitindo aos programadores escolher uma colecção de serviços e aplicações que satisfaça as suas necessidades.

4.2.1 A camada plataforma

A camada plataforma é também conhecida como a camada central do JXTA e disponibiliza os elementos absolutamente essenciais para todas as soluções P2P. Os elementos da camada central estão enumerados abaixo e são idealmente partilhados por todas as soluções P2P.

- *Peers*
- Grupos de *peers*
- Transporte de rede (*pipes, endpoints, messages*)
- *Advertisements*
- *Entity naming* (identificadores)
- Primitivas de segurança e autenticação
- Protocolos (descoberta, comunicação, monitorização)

Os primeiros seis elementos são discutidos na secção 5.3, enquanto que os protocolos são descritos na secção 5.4. De notar que os seis principais protocolos do JXTA são implementados como serviços, mas localizados na camada plataforma e desenhados como serviços centrais para distingui-los das soluções serviços da camada serviço. A camada central do JXTA é o centro fundamental na solução JXTA. Todos os outros aspectos da solução P2P JXTA nas camadas serviço e aplicação são feitos nesta camada para fornecer funcionalidade.

Os *peers* JXTA criam uma rede virtual onde qualquer *peer* pode interagir directamente com outros *peers* e recursos, mesmo quando alguns *peers* e serviços se encontram por trás de *firewalls* e *Networks Address Translation* (NATs) ou estão em transportes de rede diferentes. A comunicação através de uma *firewall* ou NAT é solucionada de duas maneiras na implementação referência. Pelo menos um *peer* do grupo dentro da *firewall* deve estar informado de, pelo menos, outro *peer* fora da *firewall*, e ambos devem suportar transporte HTTP. Além disso, a *firewall* deve permitir transferências HTTP, mas estas não necessitam estar restritas ao porto 80. A outra solução é que o nó JXTA pode ser configurado para comunicar através de um servidor de proxy HTTP e, assim, os *peers* passam a ser *routers* para tráfico P2P entre os outros *peers* do grupo. Múltiplos *peers* podem encaminhar mensagens P2P do JXTA através da *firewall*. Hoje em dia, os parâmetros para suportar *firewalls* e transporte HTTP são feitos em ficheiros de configuração mas esperam-se, no futuro, tratamentos mais avançados. [JXTAfaq]

4.2.2 A camada serviço

A camada serviço inclui os serviços de rede, tanto comuns como desejáveis no ambiente P2P, mas podem não ser absolutamente necessários para uma rede P2P operar. Exemplos de serviços são a pesquisa de recursos e *peers*, partilha de documentos entre *peers*, autenticação de *peers*, e serviços *Public Key Infrastructure* (PKI).

Os serviços na rede JXTA são publicados, descobertos e invocados por cooperação e comunicação entre *peers*. Dois níveis de serviços são reconhecidos pelos protocolos JXTA:

- Serviços de *peer*
- Serviços de grupos de *peers*

Um serviço de *peer* é acessível apenas no *peer* que publica o serviço e se esse *peer* falha, também o serviço falha. Os serviços fornecidos pelos grupos de *peers* são compostos por uma colecção de instâncias dos serviços que estão a correr nos diversos membros do grupo *peer*. Como resultado, o serviço do grupo de *peers* não é afectado se um *peer* falha, uma vez que continua disponível por outro *peer* membro.

Os serviços podem ser construídos pela comunidade JXTA ou pela equipa de projecto JXTA. Os serviços construídos no topo da plataforma JXTA fornecem capacidades específicas que são requeridas por uma variedade de aplicações P2P e podem ser combinadas para formar uma solução P2P completa.

4.2.3 A camada aplicação

A camada aplicação disponibiliza aplicações P2P comuns como o *instant messaging*, sendo construída nas capacidades da camada serviço. Algumas vezes, é difícil determinar o que é que constitui uma aplicação e o que é que constitui um serviço, dado que uma aplicação pode conter um único serviço ou agregar vários serviços. Uma aplicação é normalmente indicada quando apresenta um interface com o utilizador. Um exemplo é a *shell* JXTA, uma aplicação construída no topo do *Peer Comands,* que é um serviço. Assim, a *shell* JXTA está algures entre a fronteira da aplicação/serviço.

Aplicações no JXTA incluem aquelas construídas pela comunidade JXTA e pela equipa do projecto JXTA que maioritariamente contribuem com aplicações de demonstração como a *shell* do JXTA.

4.3 Terminologia e conceitos

A especificação JXTA define um número de conceitos comuns às redes P2P e, como resultado, temos as componentes primárias da plataforma JXTA. Esta secção dará um resumo desses conceitos.

4.3.1 Peer

Peers são nós na rede P2P constituindo a unidade de processamento central de qualquer solução P2P. Podem-se manifestar sob a forma de um processador, um processo, uma máquina ou um utilizador e podem incluir sensores, telefones, PDAs, bem como PCs, servidores e supercomputadores. Podem ser, também, aplicações distribuídas sobre várias máquinas. Os *peers* são capazes de comunicar usando protocolos requeridos por um *peer* mas não necessitam de perceber todos os protocolos JXTA (ver secção 5.4). Um *peer*, ainda assim, pode operar a um nível reduzido se não suportar um protocolo. ([JXTASpec], [Wilson2002])

Cada *peer* realiza algum trabalho útil mas fá-lo independentemente da sua tipologia. Existem três tipos de *peers* e definem um conjunto de responsabilidades em relação à rede P2P como um todo. Os diferentes tipos de *peers* são [Wilson2002]:

- ***Peers* simples:** É o tipo mais simples de *peer* na rede P2P, disponibiliza e consome recursos de outros *peers* da rede. Não é responsável por reencaminhar mensagens em benefício de outros *peers*, ou fornecer informação de terceiros à rede.

- ***Peers rendezvous*:** Disponibilizam aos *peers* simples uma maneira de descobrirem outros *peers* e *advertisements* na rede P2P.

- ***Peers routers*:** Disponibilizam serviços de encaminhamento para permitir a *peers* numa rede interna dentro de uma *firewall* e equipamento NAT, participar numa rede P2P.

4.3.2 Grupos de peers

É uma colecção de *peers* que têm um conjunto comum de interesses [JXTASpec]. Os *peers* auto-organizam-se em grupos, e cada grupo tem uma única identificação. Os protocolos JXTA descrevem como é que cada *peer* pode publicar, descobrir, juntar e monitorizar grupos de *peers*.

Antes, os conhecimentos para introduzir grupos de *peers* consistia na divisão do espaço da rede, uma necessidade que surgiu quando todos os *peers* eram capazes de comunicar usando o mesmo protocolo. As primeiras soluções P2P eram especializadas e proprietárias, e os seus protocolos associados dividiam o uso do espaço da rede de acordo com a aplicação, i.e., para partilha de ficheiros usava-se o Gnutella, para *instant messaging*, o ICQ, etc. Os grupos de *peers* dividem a rede P2P em grupos com objectivos comuns que são baseados nos elementos da lista abaixo. [Wilson2002]

- **Objectivo:** Os *peers* tipicamente formam-se e auto-organizam-se baseando-se no interesse mútuo dos *peers*. Os grupos permitem o estabelecimento de um domínio local de especialização. Os grupos de *peers* podem, por exemplo, ser divididos com base na aplicação que usam para colaborar como um grupo, ou porque querem trocar serviços apenas entre os membros do grupo e não com toda a rede P2P.

- **Segurança:** Os grupos criam um domínio local de controlo em que existe uma política específica de segurança. Assim, limitam o acesso aos recursos do grupo de *peers* através da formação de regiões lógicas sem respeito às actuais fronteiras físicas da rede.

- **Monitorização:** Os grupos de *peers* permitem aos *peers* monitorizarem um conjunto de *peers* para um propósito especial.

A especificação não determina quando, onde ou porquê se cria um grupo de *peers*, ou um tipo do grupo, ou a associação do grupo, e não limita o número de grupos a que um *peer* pode pertencer, ou se se pode formar grupos encadeados.

Um grupo de *peers* disponibiliza um conjunto de serviços onde os serviços centrais são especificados pelo JXTA, mas serviços adicionais de um *peer* do grupo podem ser desenvolvidos para entrega de serviços específicos. Os serviços centrais são:

- Serviço de descoberta
- Serviço de associação
- Serviço de acesso
- Serviço de reencaminhamento
- Serviço de resolução
- Serviço de monitorização

Os serviços são implementados num ou mais protocolos do JXTA, ou nenhum, e os grupos de *peers* não são obrigados a implementar estes serviços.

4.3.3 Transporte de rede

A troca de dados entre *peers* é tornada possível através da camada de transporte de rede, que é um mecanismo para tratar a transmissão de dados através da rede. No JXTA o conceito de transporte de rede é partido em três partes: *endpoints*, *pipes* e mensagens.

Endpoints

Os *endpoints* são interfaces de rede usados para enviar e receber dados, i.e., a fonte inicial ou o destino final de qualquer bloco de dados a ser transmitido sobre uma rede.

Pipes

Os *pipes* são canais virtuais de comunicação que são usados pelos *peers* para enviar e receber mensagens entre serviços ou aplicações sobre *endpoints*. O termo virtual refere-se ao facto de que os *peers* não necessitam conhecer os seus endereços de rede actuais para os usar. Os *pipes* são unidireccionais e assíncronos, e conectam duas ou mais aplicações, oferecendo dois modos de comunicação:

- ***Pipe point-to-point*:** Conecta exactamente dois *endpoints* de um *pipe*: um *pipe* de entrada que recebe mensagens enviadas por um *pipe* de saída.

- ***Pipe* de propagação:** Conecta um *pipe* de saída a vários *pipes* de entrada e as mensagens são enviadas a todos os *pipes* de entrada que estão à "escuta" no grupo de *peers*.

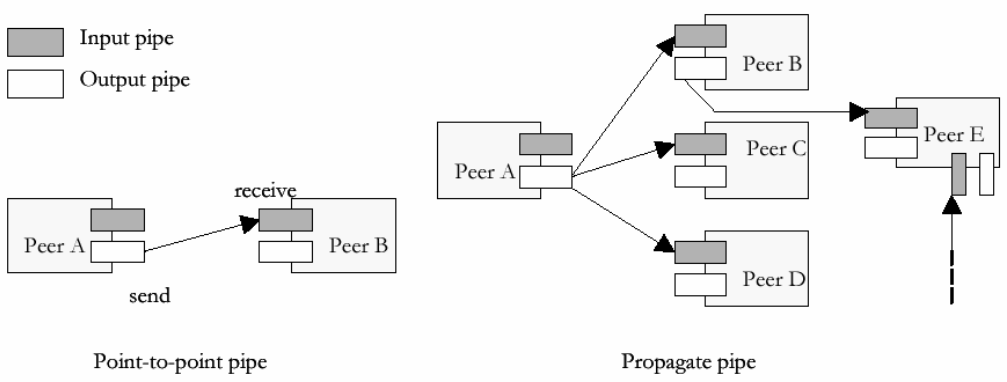


Figura 4-2 *Pipes point-to-point* e de propagação

Mensagens

A comunicação no ambiente JXTA é feita através do envio e recepção de mensagens. As mensagens são contentores de dados a serem transmitidos através de um *pipe* de um *endpoint* para outro. Para fazer o JXTA interoperável é necessário um formato *standard* para as mensagens. O JXTA usa o formato de mensagens XML, mas tanto o XML como binários podem ser enviados. As mensagens são definidas como uma sequência ordenada de contentores identificados e tipificados chamados elementos. Como resultado da escolha entre XML e binário, cada transporte JXTA pode usar o formato mais apropriado para movimentar dados.

4.3.4 Advertisements

Todos os recursos da rede, como os *peers*, os grupos de *peers*, os *pipes*, os *endpoints*, os serviços e conteúdos podem ser descritos por *advertisements*. *Advertisements* são estruturas de metadados neutros relativamente à linguagem que descrevem os recursos da rede, i.e., determinam a estrutura e representação dos dados. O projecto JXTA padroniza um conjunto de *advertisements* sendo os programadores livres de subtipificá-los para criar os seus próprios tipos. Os *advertisements* centrais são [JXTASpec]:

- *Peer advertisement*
- *Peer Group advertisement*
- *Pipe advertisement*
- *Module advertisement*
- *Peer Info advertisement*
- *Content advertisement*
- *Peer Endpoint advertisement*

Os *advertisements* são guardados (*cached*), publicados e trocados entre os *peers* para descobrir e encontrar recursos disponíveis. Os *peers* descobrem recursos através da procura dos respectivos *advertisements,* onde cada *advertisement* tem um tempo de vida que especifica a disponibilidade dos recursos associados na rede. A implementação fornece três maneiras para os *peers* descobrirem um *advertisement* [Wilson2002]:

- **Não descoberta:** Os *peers* baseiam-se na *cache* de *advertisements* previamente descobertos para disponibilizar informação dos recursos de *peers*.
- **Descoberta directa:** *Peers* da mesma LAN podem ser capazes de se descobrirem uns aos outros directamente, sem se basearem num *rendezvous* intermédio, através do uso das capacidades de *broadcast* ou *multicasting* do seu transporte de rede nativo.
- **Descoberta indirecta:** Requer o uso de um servidor *rendezvous* para agir como uma fonte de *peers* e *advertisements* conhecidos e para realização de procuras por parte dos *peers*.

4.3.5 Entidade de naming

Por forma a identificar exclusivamente os itens de uma rede P2P, como *peers*, grupos de *peers*, *pipes* e conteúdos, é necessária alguma informação para representar a exclusividade. Um *peer* usando o transporte TCP/IP pode ser identificado pelo seu endereço IP, mas o

uso de uma representação dependente do sistema é inflexível não podendo disponibilizar um sistema de identificação independente do sistema operativo ou do transporte da rede.

Qualquer dispositivo, independentemente do seu sistema operativo ou transporte de rede, deve ser capaz de participar numa rede P2P ideal. Assim, é necessário um esquema independente do sistema para tornar uma rede P2P flexível. Um ID JXTA identifica unicamente uma entidade e serve como uma forma canónica de referência a essa entidade. Os IDs são normalmente apresentados como *Uniform Resource Names* (URNs) que são uma forma de *Uniform Resource Identifiers* (URIs) que "...destinam-se a servir como identificadores de recursos persistentes e independente da localização" [JXTASpec]. Os IDs JXTA são apresentados como texto.

4.3.6 Segurança

A segurança não está implícita na estrutura do JXTA mas essa estrutura disponibiliza meios para implementar uma segurança forte usando abordagens de confiança e bem compreendidas. As mensagens XML permitem adicionar uma grande variedade de informação para mensagens, como credenciais, compilações, certificados, chaves públicas e similares. As credenciais são palavras usadas para identificar o remetente das mensagens e podem ser usadas para estabelecer chaves publicas/privadas e tecnologias de certificados digitais para implementar uma abordagem efectiva para autenticação.

O projecto JXTA fez três escolhas no que diz respeito à segurança ([JXTASec], [Yeager2002]):

- Adopção do *Transport Layer Security* (TLS) [TLS].
- Independência do transporte *end-to-end* dos protocolos JXTA.
- Uso dos certificados digitais X509.V3 onde as autoridades dos certificados centrais não são obrigatórias mas também não são excluídas.

Transport Layer Security (TLS) é um protocolo industrial para o transporte seguro de informação e uma continuação do *Secure Socket Layer* (SSL). O TLS está actualmente em desenvolvimento pela *Internet Engineering Task Force* (IETF) Network Working Group. A

implementação do TLS escolhida pelo projecto JXTA para ser incluída na distribuição JXTA é o Pure TLS [PureTLS] da Claymore Systems.

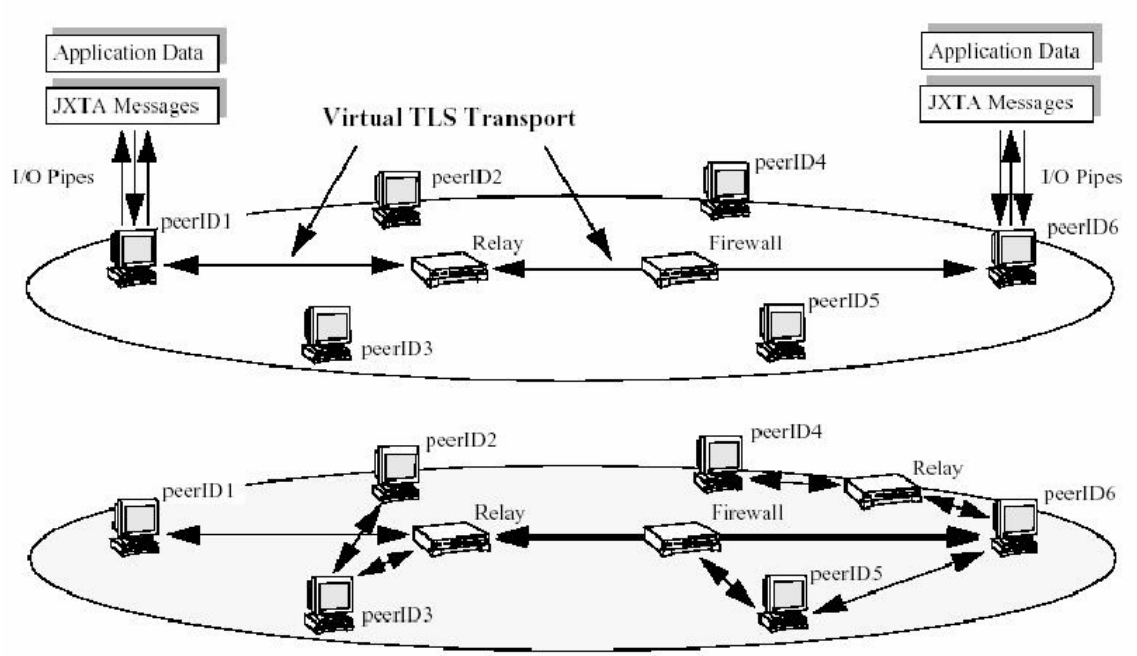


Figura 4-3 O projecto JXTA

A Figura 4.3 mostra o transporte TLS do JXTA. No nível físico, os *peers* são conectados através de *firewalls* e dos chamados *relays*. As conexões TLS criam uma camada virtual no topo da camada física onde as mensagens JXTA podem ser trocadas em segurança.

O *End-to-end transport independence* foi uma das escolhas do projecto JXTA. O JXTA é usualmente comparado ao TCP/IP que liga os nós da *Internet* onde a tecnologia JXTA conecta nós *peers* uns aos outros. Outros *standards* de transporte podem ser usados para transportar as mensagens JXTA, como o TCP/IP, sem impacto nas mensagens JXTA, uma vez que estas têm formatos predefinidos e podem incluir vários campos de dados. A importância disto é que conteúdo encriptado permanecerá sempre encriptado, independentemente de conversões de protocolo que possam ocorrer entre redes.

Os certificados digitais são usados no JXTA para garantir as identidades das partes de uma transacção e preencher os requisitos para autenticação e não repudiação. Os certificados digitais são emitidos por um parceiro de confiança, i.e., uma autoridade aderente ao

certificado X509.V3 (CA). Mas um CA centralizado nem sempre é apropriado em computação *peer-to-peer*, uma vez que as partes envolventes querem ser capazes de conduzir uma transacção segura sem envolvimento de uma estrutura centralizada. A solução JXTA permite aos próprios *peers* serem as autoridades certificadas, gerando o seu próprio certificado raiz que verifica se eles são associados com uma chave pública específica. O modelo de segurança do JXTA permite aos *peers* a formação de grupos de *peers* onde um membro pode ser designado como o CA para esse grupo. Mas um grupo de *peers* pode também usar uma autoridade certificada conhecida para emissão dos certificados.

4.4 Os Protocolos

Os protocolos são necessários em todas as trocas de dados para determinar que dados são enviados e quando. Eles estruturam a troca de informação entre os participantes usando regras acordadas por todas as partes. Os protocolos em P2P são necessários para definir todos os tipos de interacção permitida por um *peer* como parte da rede P2P. Isto inclui:

- Encontrar *peers* na rede
- Encontrar os serviços disponibilizados por um *peer*
- Obter informação sobre o estado de um *peer*
- Invocar um serviço de um *peer*
- Criar, juntar e abandonar grupos de *peers*
- Criar conexões de dados para *peers*
- Encaminhar mensagens para outros *peers*

Os *advertisements* simplificam os protocolos requeridos para fazer o P2P funcionar, uma vez que determinam a estrutura e representação dos dados e apenas deixam ao protocolo a organização da troca de *advertisements*. Os seis protocolos no JXTA definidos actualmente são:

- *Peer Discovery Protocol* (PDP)

- *Peer Information Protocol* (PIP)
- *Peer Resolver Protocol* (PRP)
- *Pipe Binding Protocol* (PBP)
- *Endpoint Routing Protocol* (ERP) / *Peer Endpoint Protocol* (PEP)
- *Rendezvous Protocol* (RVP)

Todos os protocolos são desenhados para serem fáceis de implementar em ligações unidireccionais e transportes assimétricos. A intenção foi tornar os protocolos o mais penetrantes possíveis e fáceis de implementar em qualquer transporte. O JXTA permite qualquer ligação unidireccional fazendo melhorar toda a conectividade do sistema. A comunicação bidireccional eficiente pode ser realizada pela implementação de protocolos fiáveis e transportes bidireccionais como o TCP/IP ou HTTP. [JXTAvn]

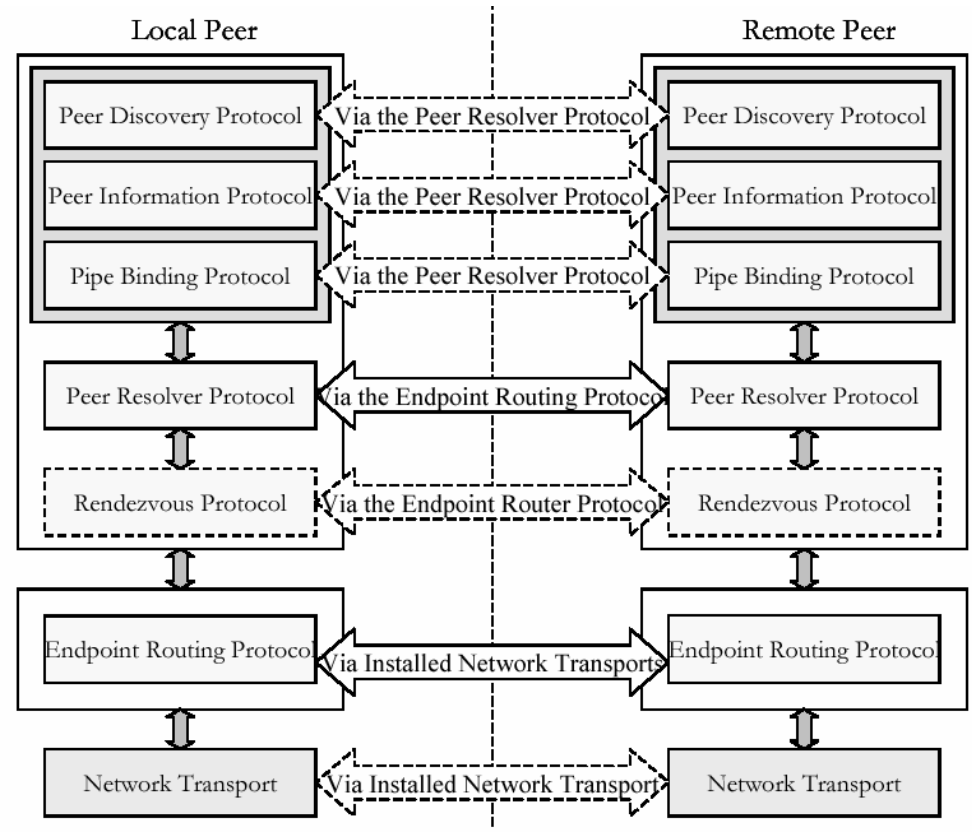


Figura 4-4 A pilha de protocolos do JXTA

A Figura 4.4 mostra a comunicação entre dois *peers* usando um protocolo e, também, como é que as camadas da pilha de protocolos são construídas no topo e "confiam" nas camadas abaixo.

Observando que para fornecer uma camada de protocolo base universal é necessário adoptar uma representação mais adequada do que a maioria das plataformas actualmente podem suportar, o XML foi escolhido para definir todos os protocolos no JXTA. Mas o JXTA não depende da codificação XML, apenas é uma forma conveniente de representação de dados. O protocolo pode usar também mensagens binárias que são mais adequadas para dispositivos sem um XML *parser*.

A responsabilidade de cada protocolo é descrita nas próximas subsecções. De notar que um *peer* apenas necessita implementar os protocolos requeridos. Por exemplo, se um dispositivo armazena todos os *advertisements* que tem em memória, o *peer* não necessita implementar o *Endpoint Routing Protocol*. Ou se não necessita obter ou fornecer informação sobre o estado, a outros *peers*, pode escolher não implementar o *Peer Information Protocol*. [JXTAvn]

4.4.1 Peer Discovery Protocol

O *Peer Discovery Protocol* (PDP) é usado pelos *peers* para descobrir todos os recursos JXTA publicados e para o *peer* anunciar os seus próprios recursos. Uma vez que todos os recursos de um *peer* são descritos usando um *advertisement*, o protocolo ajuda um *peer* a descobrir um *advertisement*. O protocolo pode ser estendido para suportar um mecanismo de procura superior mas a inclusão deste protocolo base assegura que os *peers* do JXTA podem comunicar uns com os outros ao nível mais básico.

4.4.2 Peer Information Protocol

Os *peers* usam o *Peer Information Protocol* (PIP) para obterem informação do estado de outros *peers*. Recebendo uma mensagem PIP deixa ao *peer* várias opções: pode ignorar o *ping*, enviar uma simples confirmação ou enviar uma resposta completa, que inclui o seu *advertisement*.

4.4.3 Peer Resolver Protocol

O *Peer Resolver Protocol* (PRP) faz com que um *peer* envie um pedido genérico para a procura de *peers*, grupos de *peers*, *pipes* ou outro tipo de informação para um ou mais *peers*, e receba a resposta (ou múltiplas respostas) ao pedido. Por outras palavras, o protocolo implementa um protocolo pedido/resposta, onde um pedido de um *peer* apenas pode ser enviado depois do *peer* ser descoberto via PDP. A informação do protocolo é para padronizar o formato desses pedidos. A mensagem resposta é associada através de um ID único que é incluído no corpo da mensagem. O PRP é tipicamente implementado apenas pelos *peers* que têm acesso aos repositórios de dados e que oferecem capacidades avançadas de pesquisa.

4.4.4 Pipe Binding Protocol

O *Pipe Binding Protocol* (PBP) permite a um *peer* estabelecer um canal ou *pipe* virtual de comunicação entre um ou mais *peers*, juntando dois ou mais fins de conexão de *pipes* (*pipe* de *input* e de *output*) a um ponto físico de um *peer*. O PBP é usado pelo *peer* para criação de um novo *pipe*, associação a um *pipe* existente, remoção de um *pipe* em *runtime*, e uso do PBP para envio e propagação de pedidos de *pipes*.

4.4.5 Endpoint Routing Protocol

Um *peer* usa o *Endpoint Routing Protocol* (ERP) para descobrir uma rota para uma mensagem a um *peer* destino. Dois *peers* podem não estar directamente conectados, como resultado de protocolos de transporte de rede diferentes ou separação por uma *firewall* ou NATs. O ERP é usado para determinar a informação de rota perguntando aos *peers routers* por rotas disponíveis. O próprio *peer* torna-se um *peer router* pela implementação do *Endpoint Routing Protocol*.

O ERP também é conhecido por *Peer Endpoint Protocol* (PEP).

4.4.6 Rendezvous Protocol

O *Rendezvous Protocol* (RVP) permite que um *peer* subscreva um serviço de propagação. Num grupo de *peers*, estes podem ser *peers rendezvous* ou *peers* "escutando" *peers rendezvous*. O RVP permite que mensagens sejam enviadas a todos os "ouvintes" do serviço. Por forma a propagar as mensagens, o RVP usa o PRP.

Uma versão anterior deste protocolo chama-se *Peer Membership Protocol* (PMP), e é um protocolo mais especializado cuja intenção é gerir grupos (associações), por exemplo, subscrevendo, obtendo informação e cancelando membros.

4.5 A Comunidade JXTA

JXTA começou como um projecto de pesquisa da Sun Microsystems mas, após reconhecerem que o esforço iria beneficiar os programadores fora da Sun, foi formada a comunidade JXTA sendo o *website* [JXTAws] inaugurado em Abril de 2001. A comunidade JXTA permite que todos os programadores se juntem, no esforço de desenvolverem o JXTA, e todas as especificações e implementações sejam em código aberto sobre a *Apache Software Licence*.

Os membros participantes da comunidade JXTA usam o *website* para divulgação de *papers*, *tutorials* e projectos JXTA pioneiros. Esses projectos podem ser iniciados pelos membros da comunidade JXTA, depois de aprovados pela comunidade e qualquer um que queira será bem-vindo. Os projectos podem ser divididos em seis categorias: componentes *core*, serviços, aplicações, *demos* e *tutorials*, forja (projectos JXTA em fase embrionária) e outros.

Os projectos da categoria das componentes *core* incluem a plataforma do projecto, a implementação referência do JXTA em Java e a implementação do JXTA em outras linguagens, como C, Perl, Smalltalk e Phython. Outros projectos são mais especializados para dispositivos específicos, como o JXTA para PocketPC.

O projecto mais interessante que está em conexão com esta tese é o projecto JXTA para J2ME (JXME) [JXME].

4.6 O Projecto JXTA para J2ME

JXTA para J2ME, também conhecido por JXME [JXME], é uma implementação do JXTA de modo a permitir que dispositivos integrados participem na comunidade JXTA usando J2ME e comunicando com *peers* JXTA correndo em *desktops*, *workstations* ou servidores. Desta forma, o JXTA para J2ME estende a visão de que JXTA deve correr em qualquer dispositivo.

A visão do projecto JXTA para J2ME é o fornecimento de uma infra-estrutura para redes de dispositivos auto-organizadas. Isto irá permitir o desenvolvimento de aplicações para colaboração espontânea e *ad-hoc*. Um cenário possível será a configuração espontânea de uma *Personal Area Network* (PAN) de dispositivos. Portanto, quando um PDA se aproxima de um telefone celular e/ou de um *laptop*, poderão comunicar entre si.

Quando se desenhou o JXME, diversas restrições e dificuldades impostas pelos telefones celulares e dispositivos similares tiveram que ser tidas em consideração e influenciaram as escolhas do desenho. Os telefones de hoje têm uma média de memória de 123Kb para armazenamento de MIDlets *suites*, e alguns fabricantes concedem ainda menos espaço. Além disso, o tamanho da memória de persistência e *heap* é limitada e o dispositivo pode ter latências altas em largura de banda.

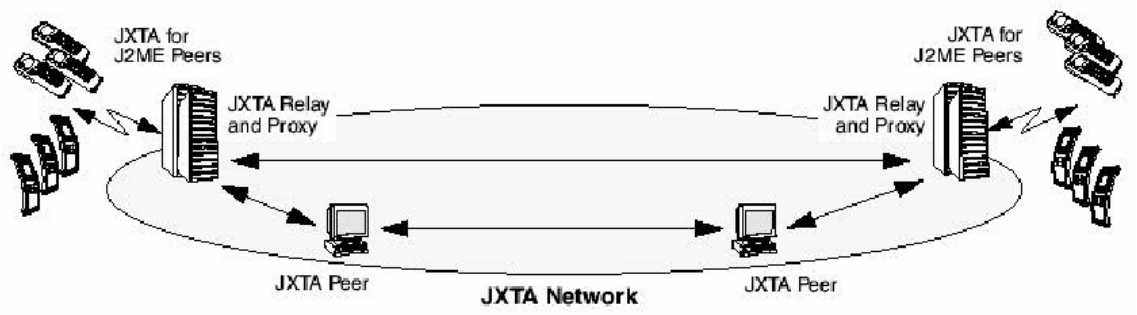


Figura 4-5 JXTA para J2ME

Existe a imposição de que os dispositivos usem a *Connected Limikted Device Configuration* (CLDC) e *o Mobile Information Device Profile* (MIDP). Esta limitação foi necessária uma vez que a implementação completa do JXTA requereu recursos que excediam os dos dispositivos integrados. Uma implementação para CDC está também em desenvolvimento e, de acordo com um dos donos do projecto, corre bem com o *Personal Profile*.

A implementação actual da plataforma JXME tem diversas restrições, e para fornecer uma comunicação *peer-to-peer* é necessário recorrer a um *relay*. O MIDP tem bibliotecas limitadas. Estas limitações são reflectidas nas capacidades do JXME. Como resultado, o JXME é implementado usando transporte HTTP. Para mensagens, JXME usa mensagens binárias que obedecem ao formato de mensagem binária do JXTA e a segurança não está contemplada. Outras funcionalidades não suportadas são o serviço *relay*, o serviço de *routing*

e o serviço *rendezvous*. Estas funções serão suportadas quando os telefones celulares amadurecerem, existir melhor suporte e o MIDP tiver bibliotecas disponíveis para suportá-las.

A intenção para o futuro é disponibilizar comunicações *peer-to-peer* sem recurso a um *relay*. Para executar os objectivos, enquanto se consideram as restrições de usar o J2ME, é necessário um servidor de *relay* para efectuar parte do trabalho. Além disso, este *relay* fará o JXTA disponível em dispositivos atrás de uma NAT. O JXME é, portanto, separável em módulos: um *peer* e um serviço *relay*.

4.6.1 O peer do JXTA para J2ME

O JXTA para J2ME irá fornecer os blocos necessários para a comunicação com o *relay* e, indirectamente, com o JXTA. A primeira versão da API foi apresentada em Março de 2002 e está em constante mudança, devendo ter um impacto mínimo (ou nenhum) nas aplicações desenvolvidas.

A API JXME disponível consiste em três classes: PeerNetwork, Message e Element. A Figura 4.6 mostra as relações entre as classes. A classe PeerNetwork é usada para comunicar com a rede JXTA e usa a classe Message para criar mensagens a enviar a outros *peers*. Uma Message consiste num ou mais Elements descrevendo os atributos através de pares nome-valor. A API está implementada de forma a que o protocolo de transporte possa ser facilmente alterado desde que a *Generic Connection Framework* do J2ME, definida no CLDC, suporte a nova tecnologia de rede.

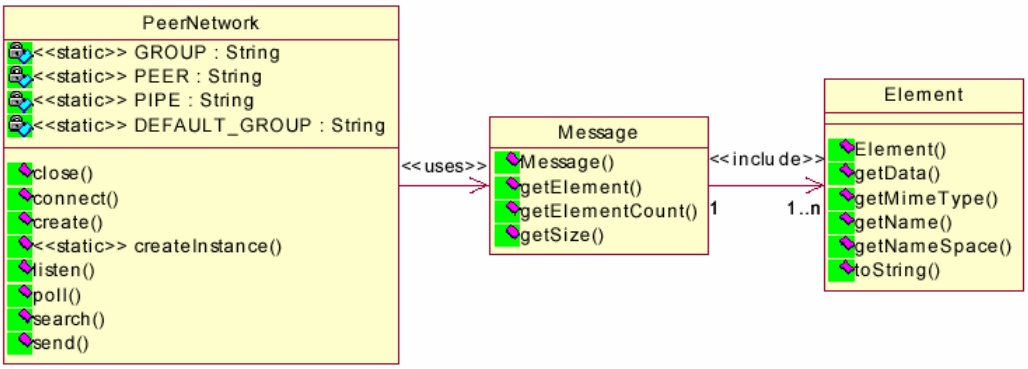


Figura 4-6 A API do JXTA para J2ME

A primeira versão do JXME suporta as seguintes funcionalidades:

- Entrega de *pipes*, grupos e *peers*
- Capacidade de criar *pipes*, tanto *point-to-point* como *pipes* de propagação e grupos
- Adição de grupos
- Comunicação com outros utilizadores JXTA através de *pipes* JXTA.

4.6.2 O Serviço relay do JXTA para J2ME

Os *peers* do JXTA para J2ME podem apenas actuar como arestas, ou *peers* simples, i.e., não podem assumir o papel de *peers* mais sofisticados que oferecem serviços a outros membros do grupo. Os *peers* por si só apenas podem realizar tarefas básicas, e as operações pesadas têm que ser feitas pelos *peers* chamados *relays* JXTA, como é mostrado na Figura 4.5.

O *relay* é parte integrante da versão Java 2 Standard Edition (J2SE) do JXTA [JavaJXTA] e corre num *peer rendezvous*. O serviço *relay* é responsável por:

- Fornecer interoperabilidade com o protocolo JXTA
- Representar o *peer* JXME na rede JXTA: o *relay* deve agir como um *proxy* para os *peers* JXME e realizar descobertas de utilizadores, grupos e *peers*. Outras tarefas são a criação de *pipes* e grupos, grupos que se pretendem juntar e comunicar. O *relay* vai ainda armazenar todas as mensagens recebidas para o *peer* JXME, e o *peer* J2ME vai periodicamente ao *relay* buscar as suas mensagens.
- Filtrar o tráfego JXTA: o *relay* irá filtrar todas as pesquisas da rede e responder em benefício do *peer wireless*.
- Optimizar os *advertisements*: o *relay* deve filtrar os *advertisements* desnecessários desde que o *peer* JXTA para J2ME não tenha memória suficiente para guardar todos os *advertisements*. As mensagens e *advertisements* recebidos são reduzidos a um mínimo, de forma a serem pequenos.
- Tradução de mensagens: os *peers* JXME usam mensagens binárias e o serviço de *relay* terá que traduzir mensagens JXTA XML em mensagens binárias e vice-versa.

Os *relays* não são uma ideia nova para o JXTA. São já usados para aceder a *peers* JXTA atrás de *firewalls* e permitem que redes usando NAT comuniquem com o mundo exterior. Os *peers* JXTA não têm que manter a relação estática com um determinado *relay* JXTA é, portanto, diferente do modelo cliente/servidor. Os *peers* JXTA podem mudar de *relay* dinamicamente, ou estar em relação com múltiplos *relays* ao mesmo tempo. Um cenário futuro é o de *peers* poderem procurar *relays* JXTA e configurar um deles para ser o *default*.

4.7 Sumário

JXTA é uma nova plataforma distribuída desenhada para resolver um diverso número de problemas na computação distribuída moderna, especialmente na área largamente falada como computação *peer-to-peer* (P2P). O objectivo é fornecer interoperabilidade entre entidades na rede, ser independente da plataforma e oferecer ubiquidade, i.e., qualquer dispositivo com "coração" digital pode participar.

A especificação JXTA define um número de conceitos que são comuns às redes P2P e como resultado temos as componentes primárias da plataforma JXTA. Os conceitos são *peers*, grupos de *peers*, *endpoints*, *pipes*, *messages*, *advertisements*, identificação de entidades e segurança. Um peer é um nó na rede P2P, enquanto que um grupo de *peers* é uma colecção auto-organizada de *peers* que têm um conjunto comum de interesses. A comunicação entre *peers*, na forma de mensagens, ocorre através de *pipes*, de um *endpoint* a outro. *Advertisements* são estruturas metadata em linguagem neutra que descrevem os recursos de uma rede, i.e., determinam a estrutura e representação dos dados. São publicados e trocados entre os *peers* para descoberta de recursos disponíveis e têm um tempo de vida que especifica a disponibilidade do recurso associado. Entidades na rede são identificadas com um ID único que é independente da rede e notações específicas de outros sistemas. A segurança é fornecida através do uso do *Transport Layer Security* (TLS) e de certificados digitais, juntando ao desenho independente do transporte que fornece segurança no transporte *end-to-end*.

A base da especificação JXTA consiste num conjunto de protocolos que são independentes da linguagem, independentes da plataforma e agnósticos à rede, i.e., não assumem nada sobre a rede. Seis protocolos estão já definidos. Os protocolos permitem a descoberta de recursos na rede, a possibilidade de obter informação sobre o *status* de outro *peer*, para juntar um *pipe* a um *endpoint* físico de um *peer*, encontrar uma rota apropriada para outro

peer, subscrever um serviço de propagação e um protocolo para efectuar *queries* genéricos de pedidos/resposta. O protocolo usa XML ou mensagens binárias para transferência de informação entre *peers*.

JXTA para J2ME, JXME, é um projecto que tem na mira o fornecimento de funcionalidades do JXTA aos dispositivos integrados, *java enabled*, *wireless*. Como resultado das limitações da rede em MIDP, a implementação é dependente de um *relay* para fazer a maior parte do trabalho. A outra parte da implementação, o *peer* JXME, fornece uma API simples que usa HTTP para comunicar com o *relay*.

5 Limitações no desenvolvimento de aplicações para dispositivos móveis

Como se sabe os dispositivos móveis são de tamanho reduzido, consequentemente com ecrãs pequenos e capacidade de processamento reduzida. Assim, quando se pensa em desenvolver aplicações para estes dispositivos é necessário ter em conta certos aspectos por forma a ultrapassar algumas limitações.

Neste capítulo, são abordados alguns desses aspectos.

5.1 Portabilidade vs Especificidade

Dada a grande diversidade de dispositivos móveis existentes no mercado, é necessário decidir se a aplicação a desenvolver será específica para um determinado dispositivo ou se, pelo contrário, essa aplicação deverá funcionar correctamente, independentemente do dispositivo onde seja executada.

Claramente, há vantagens e desvantagens em qualquer das duas abordagens.

As **vantagens** no desenvolvimento de aplicações para **dispositivos específicos** são:

- Usar uma eventual API do fabricante por forma a tirar o máximo partido das capacidades e funcionalidades desse dispositivo;
- Tamanho do código menor e mais eficaz dada a especificidade do ecrã, teclado, memória, velocidade processamento, etc.

Como **desvantagens** temos:

- Pequena abrangência de utilizadores, uma vez que existem diversos fabricantes de dispositivos móveis bem sucedidos;
- Por forma a colmatar a desvantagem anterior, torna-se necessário desenvolver várias versões da aplicação, uma para cada dispositivo. Ora, isto torna-se demasiado moroso, não só pelo facto de se ter de desenvolver especificamente para cada

dispositivo implicando conhecer muito bem as características de cada dispositivo, mas também porque sempre que há actualizações na aplicação será necessário reprogramá-las em todas as versões.

Obviamente, as vantagens e as desvantagens no desenvolvimento de **aplicações portáveis** são as opostas às desvantagens e vantagens no desenvolvimento de aplicações específicas. Vejamos as **vantagens**:

- Larga abrangência de dispositivos onde a aplicação funcione correctamente (praticamente em todos);

- As eventuais actualizações são efectuadas apenas uma vez, por existir apenas um código fonte genérico.

As **desvantagens** são:

- É difícil, se não impossível, tirar o máximo partido de cada dispositivo;

- Código maior e provavelmente menos eficaz dada a diversidade de dispositivos a ter em conta

As capacidades dos dispositivos móveis estão directamente dependentes do preço e, de certo modo, de manobras de *marketing*. Assim, o sucesso dos programadores depende da sua habilidade de criar *software* que abranja uma larga gama de plataformas e que esteja pronto a funcionar na altura em que o fabricante, o operador e o consumidor estejam preparados e desejem determinado serviço.

Os programadores são muitas vezes surpreendidos com a rapidez com que certas características se tornam disponíveis. Por ser muito difícil adivinhar quando é que um fabricante disponibiliza novas funcionalidades, o esforço do programador deve concentrar-se na criação de um alto nível de flexibilidade para as aplicações J2ME de forma a que sejam compatíveis com os dispositivos específicos.

Balancear o uso de memória, os parâmetros de armazenamento, o tamanho e características do ecrã, as especificações de rede, o tamanho da aplicação e suas características são uma tarefa obrigatória.

Nos pontos seguintes são abordados alguns aspectos que influenciam o desenvolvimento de aplicações independentes do dispositivo.

5.2 Interface com o Utilizador

Um dos grandes problemas na realização de uma aplicação para dispositivos móveis é a criação da interface com o utilizador que funcione nos diferentes dispositivos.

O problema reside no facto desses dispositivos terem ecrãs com tamanhos muito distintos, assim como, diferentes mecanismos de entrada de dados (*input*). Uns têm teclado físico, outros *touch screen*, etc.

O desafio está em desenhar uma interface que funcione em vários dispositivos e que, mesmo assim, aproveite as características de cada um ao máximo.

5.2.1 Ecrã e teclado

O tamanho reduzido do ecrã é um dos grandes problemas no desenvolvimento de aplicações para dispositivos móveis. Enquanto a resolução do ecrã e o número de cores continuam a aumentar, o tamanho dos ecrãs mantém-se porque os utilizadores não gostam de telemóveis grandes e pesados.

Figura 5-1 Telemóvel com ecrã muito pequeno

Os dispositivos tipo PDA (*Personal Digital Assistant*) geralmente têm um grande ecrã comparado com os telemóveis. Têm, normalmente, um *stylus*, uma caneta de plástico, que funciona como um rato, para apontar e pressionar um ecrã sensível ao toque. Usualmente, estes dispositivos dispõem de poucos botões físicos.

Geralmente, os telemóveis, por outro lado, têm um ecrã mais pequeno que não é sensível ao toque. Os telemóveis normais têm muitos botões, tipicamente um teclado numérico, teclas de setas e um botão de seleccionar.

As características dos telemóveis tipo PDA favorecem interfaces com o utilizador que tenham botões no ecrã (botões virtuais), uma vez que não têm teclado. O ecrã maior permite mostrar mais informação ao mesmo tempo. Um exemplo de uma interface num computador de mão é mostrado na Figura 5.2.

Figura 5-2 Telemóvel tipo PDA

Num telemóvel, o ecrã mais pequeno não permite mostrar tanta informação e este espaço tem de ser aproveitado ao máximo, pois não se deve exceder o tamanho do ecrã nem ocultar informação. Os botões deixam de estar no ecrã e passam a ser físicos como se pode ver na Figura 5.3.

Figura 5-3 Telemóvel Normal

Existem algumas teclas especiais que merecem alguma atenção e que estão definidas no MIDP [JSR-37], ajudando a manter a portabilidade entre telemóveis diferentes. Estas são chamadas teclas de códigos de acção. Esses códigos são: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, e GAME_D. Essas teclas de acção são garantidas em todos os telemóveis com capacidades J2ME. O método (função) MIDP `getGameAction(int keyCode)` devolve um código constante usando o código das teclas como parâmetro de entrada.

Quanto ao ecrã, a portabilidade também pode ser mantida através do pedido das dimensões do ecrã do telemóvel. Os métodos `getWidth()` e `getHeight()` devolvem o tamanho do ecrã. Conhecendo estes parâmetros é possível definir zonas no ecrã com valores relativos em vez de valores absolutos, trabalhando em baixo nível (Canvas) e não de componentes de alto nível predefinidos.

Figura 5-4 Mostra os diferentes métodos de portabilidade na classe Canvas

5.2.2 Cores Limitadas e ausência de suporte para som

Um grande número de telemóveis ainda são a "preto e branco", embora a maioria dos telemóveis já suporte cores. Tendo em vista a portabilidade da aplicação é necessário fazer uma boa escolha das cores a usar, devendo ser consideradas as cores contrastantes entre si, de forma que "funcionem" tanto em dispositivos coloridos como em monocromáticos ou com tons de cinzento.

Embora estes dispositivos estejam normalmente dotados de som, as aplicações têm uma capacidade limitada para usar essa capacidade do dispositivo. A especificação J2ME não requer que os telemóveis suportem som sequer, embora as primitivas de Java permitam aos telemóveis o uso de alguns sons, especificamente sons de alerta. Apesar do suporte MP3 e MIDI estar a aumentar, geralmente, apenas uma voz e um canal são possíveis.

5.3 Tamanho limitado para a aplicação

A maioria dos telemóveis com capacidade Java dispõe de uma quantidade de memória [KnudMem] limitada para as MIDlets.

Além disso, existe também limitação no tamanho da MIDlet em si. Este limite depende do fabricante e das suas políticas. Alguns fabricantes também impõem limite no

armazenamento de persistência (armazenamento de objectos criados pela aplicação). Para dificultar nota-se a falta de divulgação de informação por parte de algumas marcas.

Desta forma, desenvolver um jogo atrevido ou uma aplicação completa pode ser um grande desafio mas temos que nos lembrar que os primeiros computadores pessoais tinham 64KB de memória ou menos e, mesmo assim, as pessoas ainda se divertiam com os seus jogos. Os limites são muito menos apertados para os telemóveis recentes, como o Nokia 3650, que pode correr aplicações com vários Mbs. Por isso, qual é a aproximação mais eficiente para tornar portável uma aplicação para múltiplos dispositivos?

5.4 Sumário

Neste capítulo pode-se constatar que desenvolver aplicações para dispositivos móveis não é tarefa fácil dadas as suas limitações. Entre estas, destacam-se a enorme variedade de tamanhos de ecrãs, diversidade de suporte de cores, com teclado físico ou com *touch screen*, limitações no tamanho final que a própria aplicação pode ocupar, memória disponível e espaço para armazenamento.

É necessário trabalhar a baixo nível e empregar todos os expedientes necessários para ir resolvendo das dificuldades que vão aparecendo. Alguns destes expedientes serão descritos em detalhe no próximo capítulo onde se apresenta o protótipo desenvolvido.

6 Protótipo

Relembra-se que o objectivo desta tese de mestrado baseia-se na investigação das reais capacidades de conectividade *peer-to-peer* de dispositivos móveis, como telefones celulares e PDAs, usando J2ME e fazer a avaliação de tecnologias existentes como o JXTA para J2ME (JXME).

Foi feita uma breve introdução, no capítulo 3, ao J2ME e CLDC/MIDP, APIs do JAVA para pequenos dispositivos e focadas as capacidades disponibilizadas como parte destas plataformas. No capítulo 4, foi efectuado um resumo do JXTA, uma tecnologia distribuída.

De forma a cumprir os objectivos iniciais foi desenvolvido um protótipo para avaliar as reais capacidades de conectividade *peer-to-peer* de dispositivos móveis. Após o estudo do JXTA, mais especificamente, a sua componente JXME (JXTA para J2ME), indicava à partida ser uma aposta acertada para inclusão no desenvolvimento do protótipo de demonstração. No entanto, o JXTA não pode ser usado devido à presença de uma *firewall* no local de desenvolvimento do protótipo, tornando desta forma o JXME inoperacional. Como alternativa, optou-se pela implementação de uma solução própria, isto é, um protocolo de comunicação.

O protótipo insere-se no contexto do tradicional jogo de golfe, mais concretamente na marcação das jogadas efectuadas por cada jogador no chamado cartão de resultados (*score card*). Com este protótipo pretende-se mostrar como se podem ultrapassar as limitações quando se programa para dispositivos móveis e, mais importante, como partilhar informação entre *peers* que, neste caso, são os jogadores de golfe. O objecto de partilha será, neste contexto, os dados de campos de golfe (ver secção seguinte).

mGolfScoreCard é o nome do protótipo desenvolvido no âmbito desta dissertação (o 'm' é de *mobile*). Digamos que esta aplicação poderá substituir a forma como, normalmente, se registam as ocorrências num jogo de golfe.

6.1 Cartão de golfe em papel (tradicional)

Um cartão de golfe para um determinado campo, usualmente, contém o nome do campo e o número de buracos e, para cada buraco, a distância, o *stroke index* e o par. No decorrer do jogo são registados, por um dos jogadores – o *marker*, também no cartão, os resultados alcançados por cada jogador em cada buraco (ver Figura 6-1). No final do jogo é necessário efectuar os cálculos que podem ser um pouco complexos (se houver jogadores com *handicaps* distintos), principalmente, porque são feitos no pequeno cartão de papel. No Anexo A pode-se encontrar um breve resumo das regras do golfe e dos termos técnicos.

Wyboston Lakes Golf Course **Score Card**

S.S.	
White	69/70
Red	72/73

Player A Handicap Allowance

Player B Handicap Allowance

Date Competition

MARKER	HOLE	WHITE	YELLOW	PAR	S.I.	RED	PAR	S.I.	PLAYER A	PLAYER B	POINTS
	1	503	486	5	6	473	5	6			
	2	357	341	4	8	319	4	8			
	3	133	110	3	14	107	3	16			
	4	393	384	4	2	365	4	2			
	5	147	139	3	18	134	3	18			
	6	280	266	4	16	261	4	14			
	7	173	163	3	10	157	3	12			
	8	362	343	4	4	349	4	4			
	9	331	316	4	12	316	4	10			
	OUT	2679	2548	34		2481	34				
	10	447	433	4	5	418	5	13			
	11	427	414	4	3	417	5	7			
	12	358	342	4	7	327	4	5			
	13	154	130	3	17	141	3	17			
	14	416	392	4	1	414	5	1			
	15	334/493	321/470	4/5	9	319/433	4/5	11			
	16	168	150	3	15	153	3	15			
	17	481	476	5	11	455	5	3			
	18	332	314	4	13	315	4	9			
	IN	3117/3276	2972/3121	35/36		2959/3073	38/39				
	OUT	2679	2548	34		2481	34				
	TOTAL	5796/5955	5520/5669	69/70		5440/5554	72/73				
								Handicap			
								Nett			

Markers Signature _____

Players Signature _____

Points/Par Result

Figura 6-1 Exemplo de um cartão de golfe convencional

6.2 mGolfScoreCard

O protótipo desenvolvido, mGolfScoreCard, além de suportar as funcionalidades de um cartão tradicional e efectuar os cálculos das pontuações finais, permite ainda a partilha de dados relativos a campos e a jogos. Por exemplo, quando alguém, possuidor de um telemóvel, for jogar golfe num campo pela primeira vez, terá que ter os dados desse campo inseridos na aplicação. Assim, ou os insere manualmente, tornando-se numa tarefa morosa, ou os obtém através de ligações *peer-to-peer* com outros dispositivos móveis (ou não) que sejam detentores dos dados requeridos, fazendo então o *download* dos mesmos. Mais ainda, no final do jogo, o jogador que fez as marcações das pontuações pode partilhá-las com os seus parceiros/adversários (ver Anexo D – Manual do Utilizador).

6.3 Soluções implementadas

6.3.1 Tamanho do ecrã

A frase é célebre e, provavelmente, do conhecimento da maioria dos informáticos: *"Write Once, Run Everywhere"*. O Java foi desenhado para ser portável através de diferentes plataformas e sistemas operativos, mas com a chegada do J2ME veio um novo pequeno entrave: "Para diferentes tamanhos de ecrã, estava pensado?" A resposta é: "Sim. E facilmente programável usando os componentes de alto nível do MIDP." Mas será a melhor escolha para estes dispositivos? De facto, para aplicações minimamente profissionais o uso dos componentes de alto nível do MIDP estará fora de questão. Se imaginarmos numa aplicação um pequeno ecrã de configuração com *radio buttons*, *checkboxes* e alguns botões, facilmente nos apercebemos que, em dispositivos com ecrãs de pequenas dimensões, inevitavelmente teriam que existir *scrolls* para se aceder a todos os campos do suposto formulário de configuração. Ora, isto provocaria uma interacção muito pouco prática se a aplicação fosse usada no dia-a-dia.

A solução mais adequada, e que foi utilizada no protótipo, será recorrendo, não aos componentes de alto nível, mas programando a baixo nível, isto é, ao nível do `Canvas`. Deste modo, usa-se toda a área gráfica disponibilizada por cada dispositivo. Essa área é subdividida em pequenas áreas formando unidades de interacção (ver o ponto 6.3.4 mais à frente).

Um telemóvel com um ecrã de tamanho mínimo e um telemóvel com um ecrã generoso, como fazer com que a aplicação funcione para os dois?

Figura 6-2 Dois telemóveis com ecrãs de tamanhos diferentes

A solução usada para a implementação foi a de dimensionar as áreas da aplicação em função da altura e largura do ecrã (usando as funções `getWidth()` e `getHeight()`).

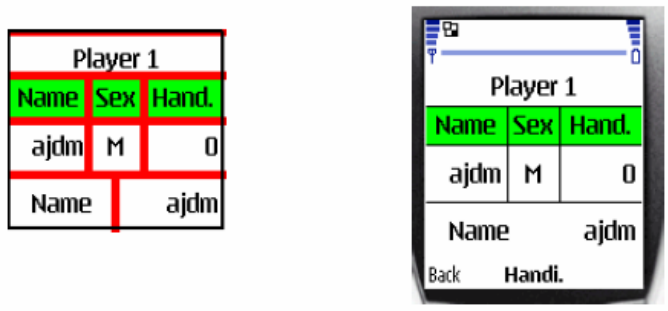


Figura 6-3 Divisão das áreas do ecrã

Como se pode ver na Figura 6-3, o espaço de cada área é determinado através do tamanho do ecrã, altura e largura, sendo, assim, possível desenhar a interface pretendida em vários telemóveis. Cada subárea é calculada de forma relativa ao tamanho total do ecrã. Para um telemóvel com *touch screen*, usualmente com ecrãs mais generosos, é criada uma área suplementar dedicada ao teclado.

6.3.2 Teclado

Teclado físico e teclado virtual

Na Figura 6-4 à direita, pode-se ver o ecrã de um telemóvel normal. Neste caso, todo o ecrã vai ser usado para mostrar informação da aplicação, já que o espaço é muito reduzido e existe o teclado físico.

Figura 6-4 Dois tipos de ecrãs diferentes, um deles com *touch screen*

Na Figura 6-4 à esquerda, pode-se ver o ecrã de um telemóvel tipo PDA. Este tipo de telemóveis costuma ter um ecrã de dimensões generosas e, ao contrário dos outros telemóveis, não tem teclas ou tem poucas e específicas. De maneira a que aplicação tenha um aspecto semelhante em todos os telemóveis, a área da aplicação é a mesma para ambos

telemóveis, sendo que para o PDA é usado um espaço extra do ecrã para o teclado virtual. A verificação é feita usando a função `hasPointerEvents()`.

Teclados virtuais e entrada de dados

Nos dispositivos com teclado físico consegue-se ter um conjunto aceitável de caracteres disponíveis para entrada de dados. Para isso, cada tecla pode servir para escrever mais do que um carácter.

Ora, nos dispositivos com *touch screen*, onde usualmente se verifica a ausência de teclas físicas, decidiu-se recorrer à mesma técnica: uma tecla para vários caracteres. Foi, então, necessário implementar um sistema que, através dos eventos e toques no ecrã, se determinasse o valor de entrada. Por exemplo, várias letras são introduzidas usando a mesma tecla. Assim, para determinar a letra escolhida, é necessário calcular o número de toques efectuados nessa tecla e ter em conta o tempo entre toques.

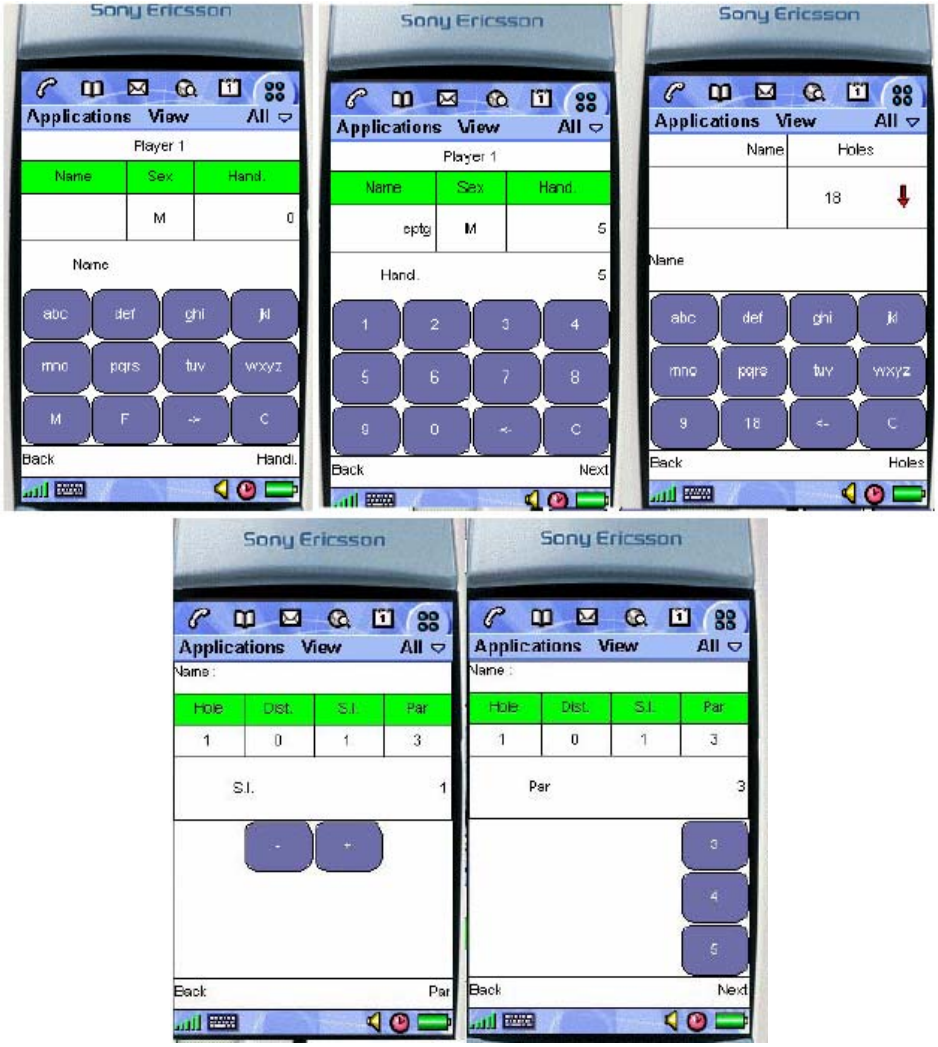


Figura 6-5 Várias possibilidades do teclado virtual

Outros aspectos da portabilidade

Então, na aplicação, porque é que a tecla C não funciona? Bem, o J2ME não define a tecla C como especificação e, como se pode ver na Figura 6-6, alguns nem têm tecla C.

Assim, a maneira de se ter teclas com a função "apagar" e que existam em todos os telemóveis, consiste em fazer uso de teclas que são constantes (existem de certeza) nos telemóveis com suporte para J2ME : o * e o #. Por exemplo, * apaga tudo e # apaga carácter a carácter.

Figura 6-6 Telemóvel sem tecla C e telemóvel com tecla C

Em alguns telemóveis, existe uma funcionalidade especial, que pressionando o botão do meio é equivalente a pressionar o *soft button* de OK. Tudo isto tem de ser tido em conta na aplicação, de maneira a que, apesar da portabilidade, se aproveitem as características dos telemóveis ao máximo. Foi necessário código especial para proporcionar aos utilizadores destes telemóveis a possibilidade de as usar na aplicação, uma vez que não estão especificadas.

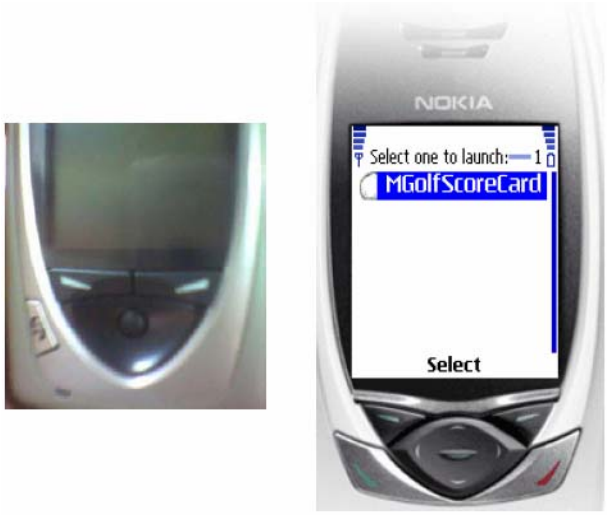


Figura 6-7 Telemóveis com funcionalidades especiais

6.3.3 Cores

Uma vez que os telemóveis não têm todos a mesma resolução, nem o mesmo número de cores disponíveis, é necessário preparar a aplicação de maneira a que as cores sejam harmoniosas e não apareça, por exemplo, um ecrã que era de um verde bonito, todo preto num telemóvel com suporte para apenas duas cores.

Figura 6-8 Várias possibilidades de cores através da programação

Nesta Figura 6-8 podem-se ver três tipos diferentes de ecrã, onde foi necessário calcular a cor através do número de cores suportado por cada dispositivo (função `numColors()` da classe `Display`), de maneira a que aplicação tivesse um aspecto mais atraente. Por exemplo, o segundo telemóvel apenas tem um conjunto limitado de tons de cinzento com que se pode contar para criar uma interface mais apelativa.

6.3.4 Língua a utilizar

O público alvo da aplicação é sem dúvida um público de língua inglesa, pois é a língua base dos praticantes de golfe. A limitação do tamanho da aplicação não permite que se possa utilizar a internacionalização pois isso aumentaria o tamanho da aplicação limitando, assim, a portabilidade.

6.4 Interface com o utilizador

Para um melhor aproveitamento do espaço e controlo do posicionamento de texto e imagens foram desenvolvidos ecrãs principais de introdução de dados através de classes de baixo nível. Esses ecrãs dividem-se, essencialmente em duas categorias: ecrã de alerta e ecrã tipo *wizard*. Existem também ecrãs de menu mas são construídos utilizando a classe de alto nível `List`.

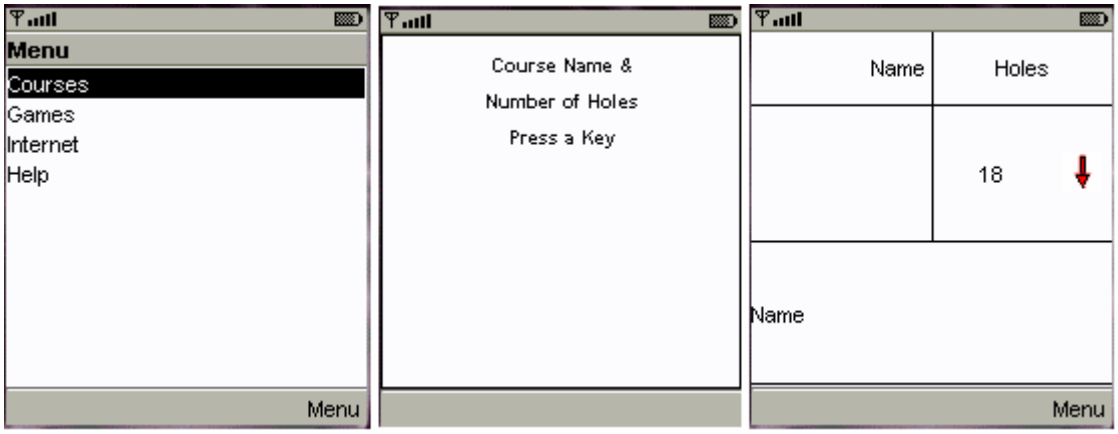


Figura 6-9 Exemplos de ecrãs: menu, alerta e *wizard*

Todos os ecrãs da aplicação e respectivo esquema de navegação podem ser consultados no Anexo B.

De forma a implementar tais ecrãs foram construídas as classes `AlertMG`, `SubCanvas`, `NCCanvas`, `GCanvas`, `NDCanvas`, `PICCanvas`, `PSCCanvas` e `SCFCanvas`.

- `AlertMG`: serve para simular um alerta
- `SubCanvas`: permite criar pequenas áreas funcionais subdividindo o Canvas
- `NCCanvas`: permite a inserção do nome e nº de buracos de um novo campo de golfe
- `NDCanvas`: complementa a classe `NCCanvas` permitindo a inserção do nome, distância, *stroke index* e par de um buraco de um novo campo de golfe

- `PSCCanvas`: permite a inserção de um novo jogo de golfe sendo pedidos a identificação do jogo e o nº de jogadores
- `PICanvas`: permite a inserção dos nomes, sexo e *stroke index* dos jogadores
- `GCanvas`: permite a marcação das ocorrências num jogo de golfe
- `SCFCanvas`: permite a amostragem dos resultados finais de um jogo

De seguida serão descritas com mais pormenor essas classes.

Classe AlertMG

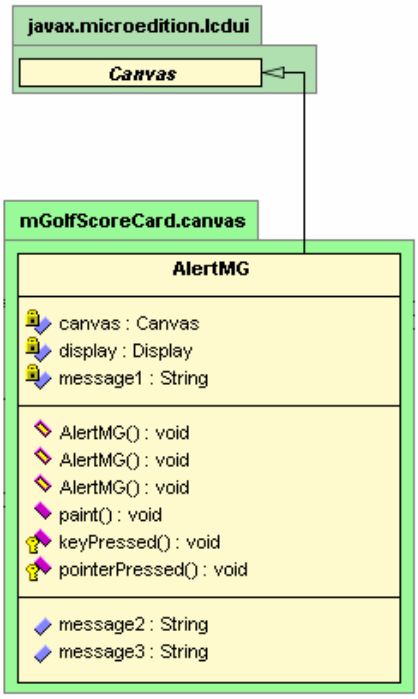


Figura 6-10 Estrutura da classe AlertMG

Classe que permite simular um alerta. Apesar de existir uma classe no J2ME com características semelhantes, não pode ser usada como primeiro ecrã em alguns telemóveis, como em muitos casos era pretendido.

Na classe `AlertMG`, o ecrã é visualizado até que se carregue num botão ou se toque no ecrã. Existem três zonas de texto e a última serve para informar o utilizador do que deve fazer.

Figura 6-11 Visualização do efeito provocado pela classe AlertMG

Classe SubCanvas

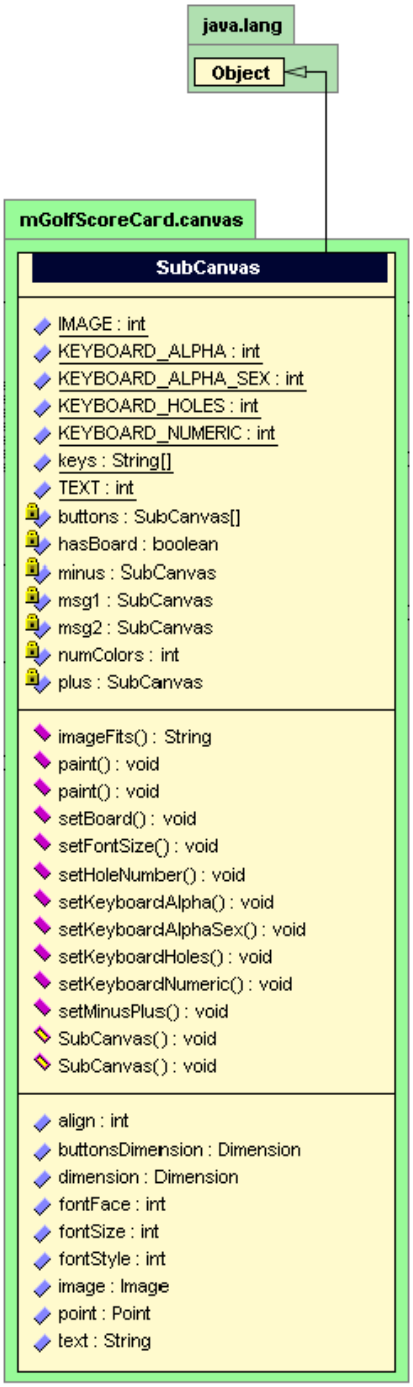


Figura 6-12 Estrutura da classe SubCanvas

Classe essencial para o desenho das outras classes que derivam de `Canvas`. Por definição, um *canvas* ocupa todo o ecrã do dispositivo, tornando-se necessário dividir o ecrã em várias

áreas funcionais recorrendo a instâncias da classe `SubCanvas`. Com esta classe é possível desenhar desde texto a imagens, assim como os teclados virtuais.

Esta classe acaba por ser a base das outras classes e o corpo para os teclados virtuais. Com ela conseguiu-se garantir a portabilidade para o maior número de dispositivos.

Classe NCCanvas

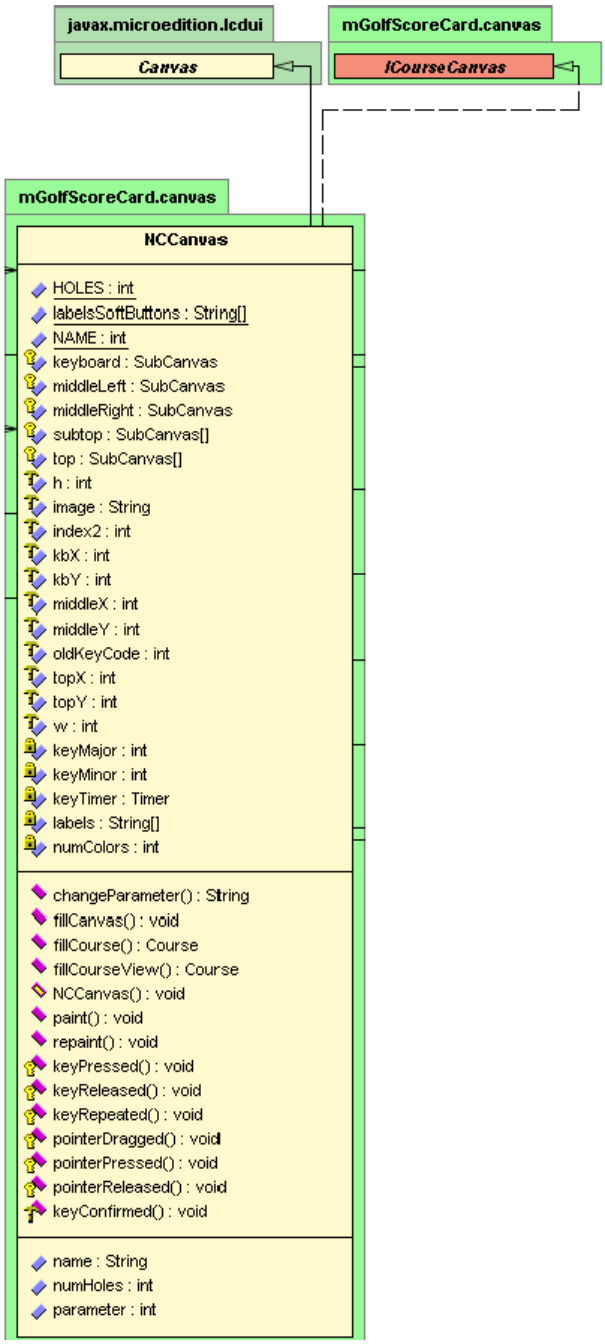


Figura 6-13 Estrutura da classe NCCanvas

Classe que serve para mostrar um ecrã de introdução de dados e garantir que estes sejam guardados, pois implementa a interface `ICourseCanvas` que foi desenvolvida de

maneira a garantir que a informação do campo introduzida neste ecrã seja guardada. De seguida podem ver-se duas imagens que ajudam a compreender isso mesmo.

Figura 6-14 Visualização do efeito provocado pela classe NCCanvas

Na classe `NCCanvas` podemos introduzir o nome através do sistema especial de escrita implementado. Também é possível alterar os valores do número do campo.

Para alterar o valor do campo não é necessário sair do ecrã do nome, basta mover as setas para cima ou para baixo. Acaba por ser uma espécie de atalho.

Classe NDCanvas

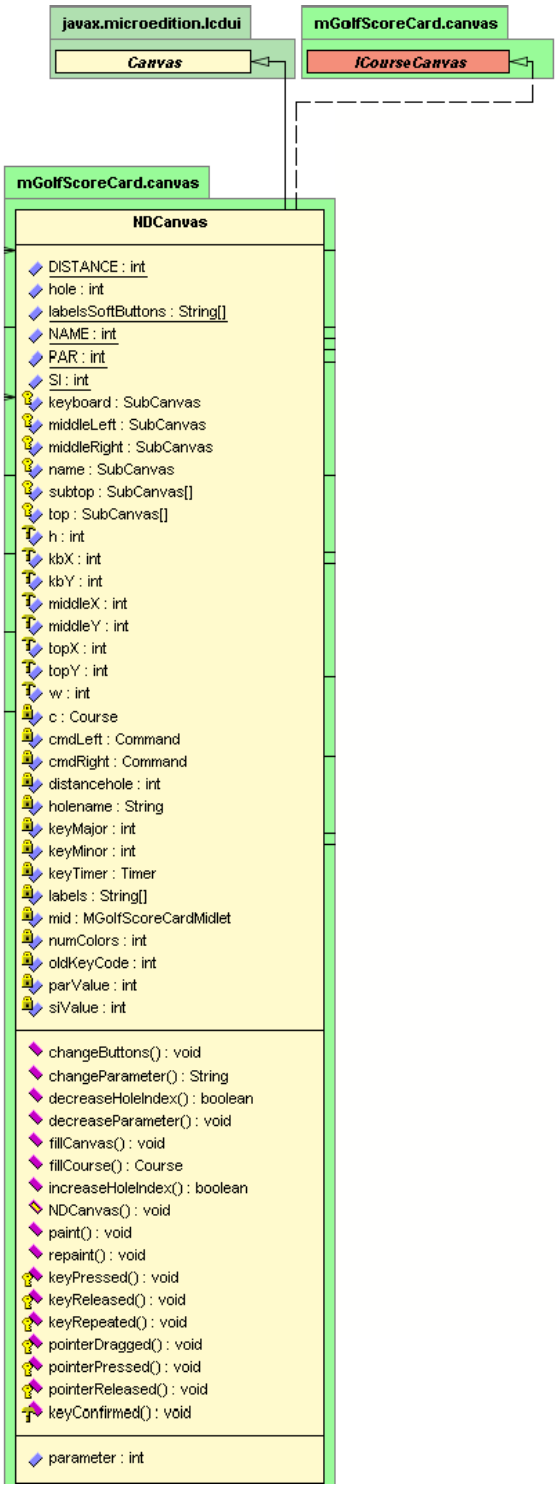


Figura 6-15 Estrutura da classe NDCanvas

Classe que serve para mostrar um ecrã de introdução dos dados e garantir que estes sejam guardados, pois implementa a interface `ICourseCanvas` que foi implementada de maneira a garantir que seja guardada a informação do campo que é preenchida neste ecrã.

Com este ecrã é possível introduzir a informação essencial para um buraco, o nome do buraco, a distância, o índice da tacada e o valor do par.

Para facilitar a introdução dos dados, este ecrã está cheio de funcionalidades implementadas. O nome e distância são simples entradas de dados, texto e números.

A introdução do valor SI (*Stroke Index*) é, nos teclados físicos, feita através das setas Cima e Baixo. A justificação é simples: não permitir a introdução de erros, pois, com as setas apenas se visualizam valores válidos. Outro aspecto está no valor do Par, os valores possíveis são apenas 3, 4 e 5 e, portanto, apenas é possível digitar nessas 3 teclas.

Outro aspecto que melhora bastante a navegação é a possibilidade de navegar neste menu com as teclas Esquerda e Direita. Nos telemóveis com *touch screen* funciona da forma como se pode ver na Figura 6-16.

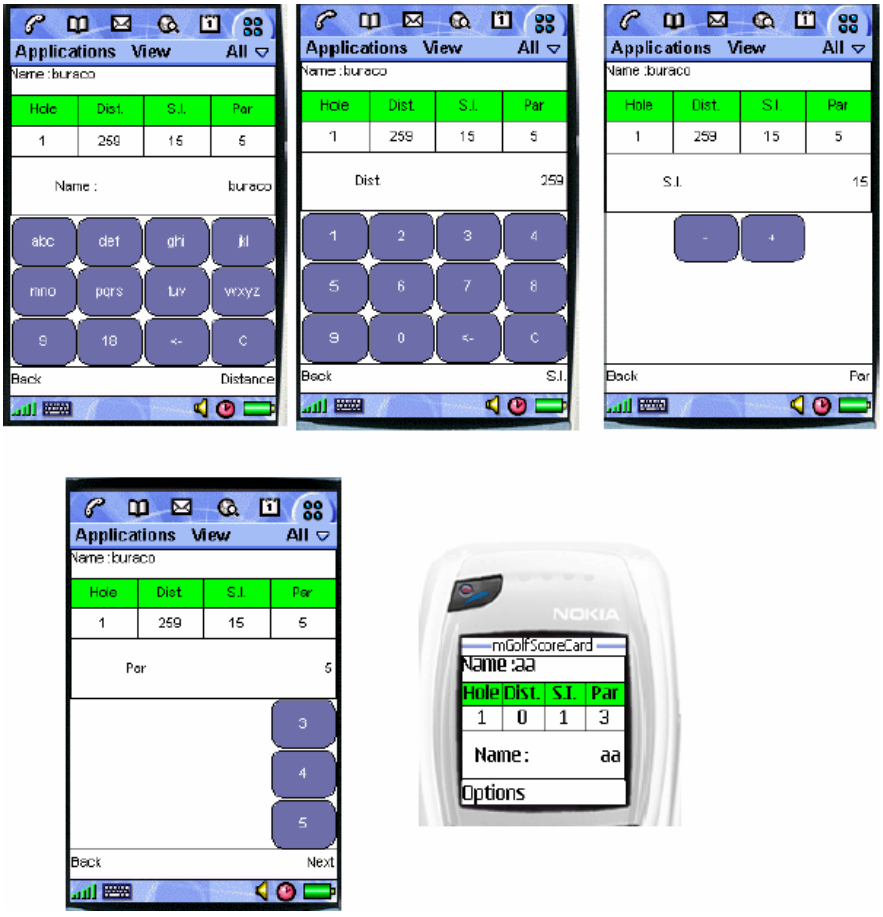


Figura 6-16 Visualização do efeito provocado pela classe NDCCanvas

Classe PSCCanvas

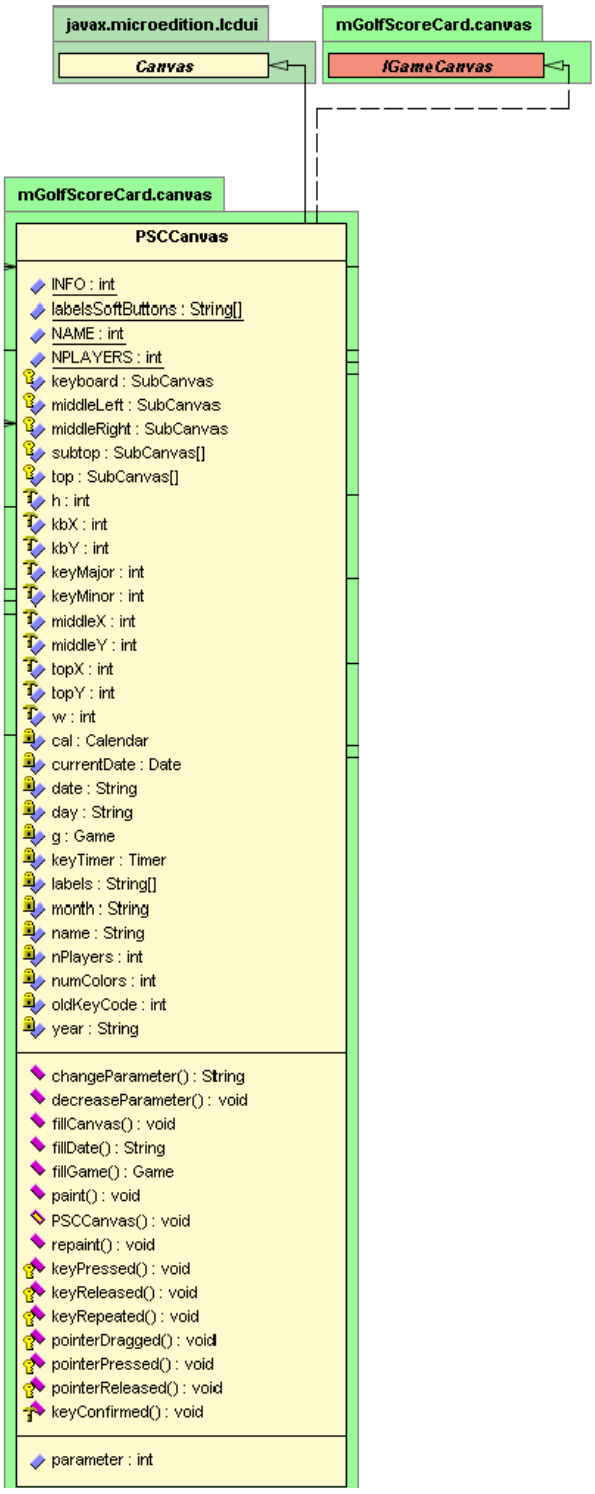


Figura 6-17 Estrutura da classe PSCCanvas

Classe que serve para mostrar um ecrã de introdução dos dados e garantir que estes sejam guardados, pois implementa a interface `IGameCanvas` que foi implementada de maneira a garantir que seja guardada a informação do jogo que é preenchida neste ecrã.

Neste ecrã, o utilizador insere um nome do jogo que identificará o jogo que irá jogar e número de jogadores. A data aparecerá automaticamente.

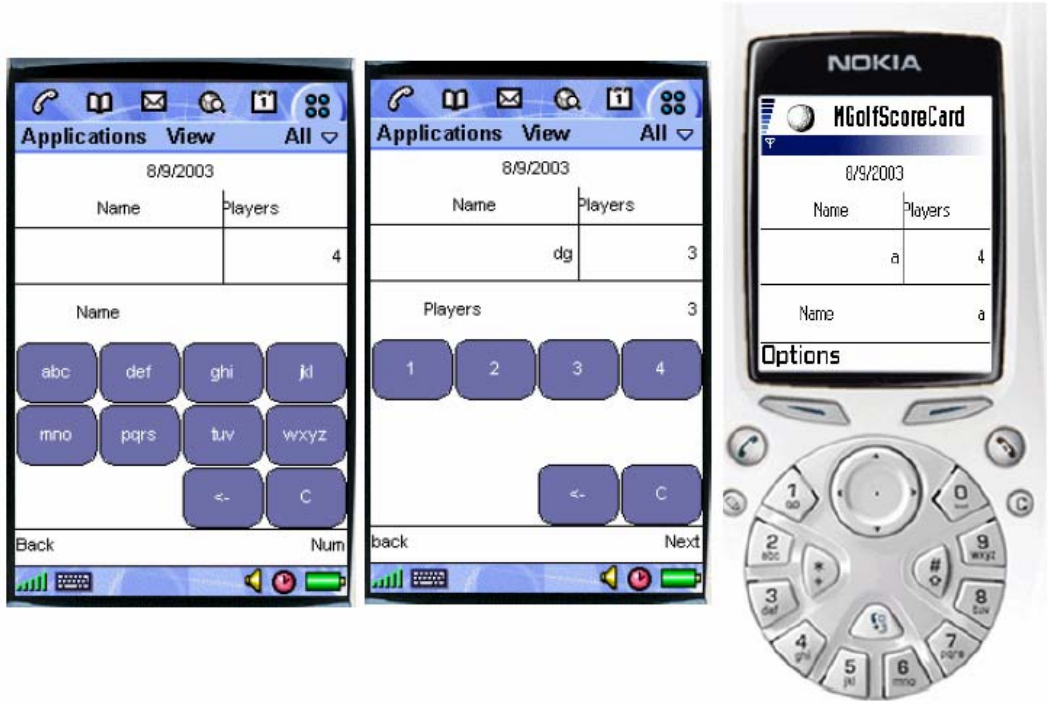


Figura 6-18 Visualização do efeito provocado pela classe PSCCanvas

Classe PICanvas

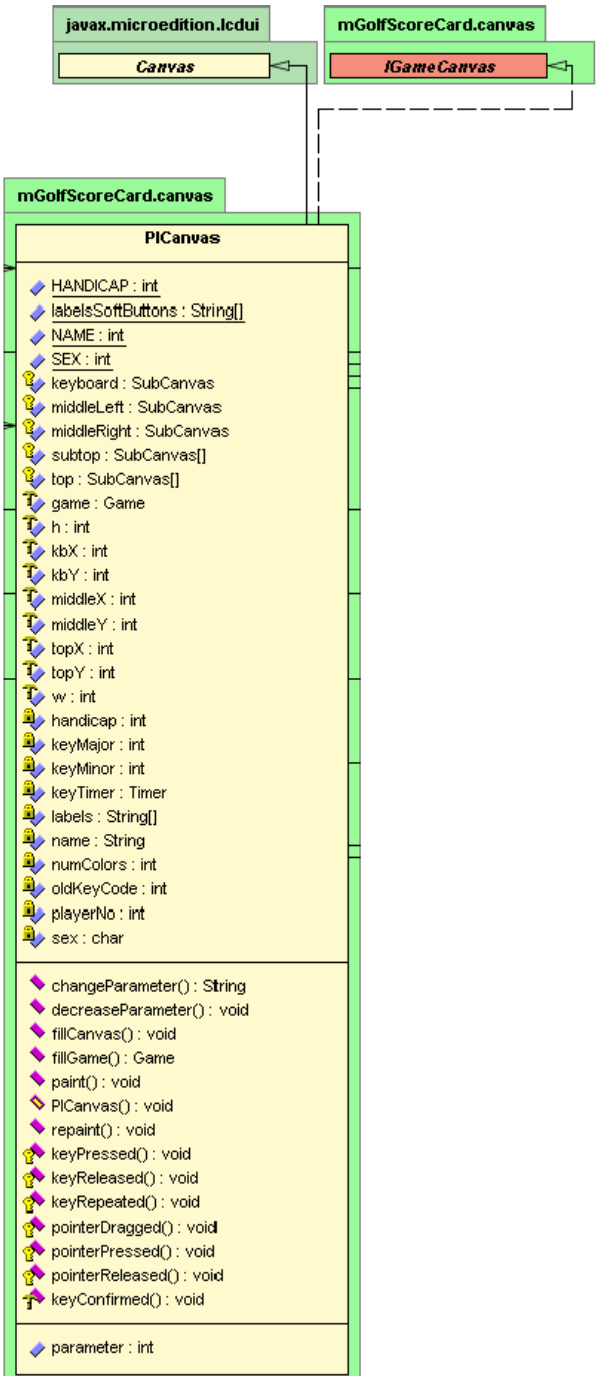


Figura 6-19 Estrutura da classe PICanvas

Classe que serve para mostrar um ecrã de introdução dos dados e garantir que estes sejam guardados, pois implementa a interface `IGameCanvas` que foi implementada de maneira a garantir que seja guardada a informação do jogo que é preenchida neste ecrã.

Com este ecrã é possível introduzir a informação de cada jogador como o nome, o sexo e o *handicap*. O parâmetro sexo é importante pois os valores máximos do *handicap* variam para mulheres e homens. Desta forma, é possível fazer uma correcta validação dos valores.

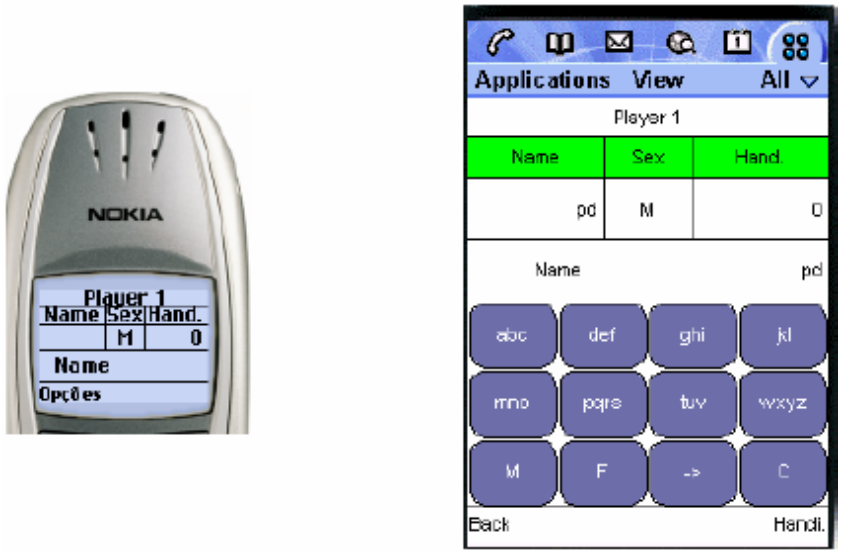


Figura 6-20 Visualização do efeito provocado pela classe PICanvas

Classe GCanvas

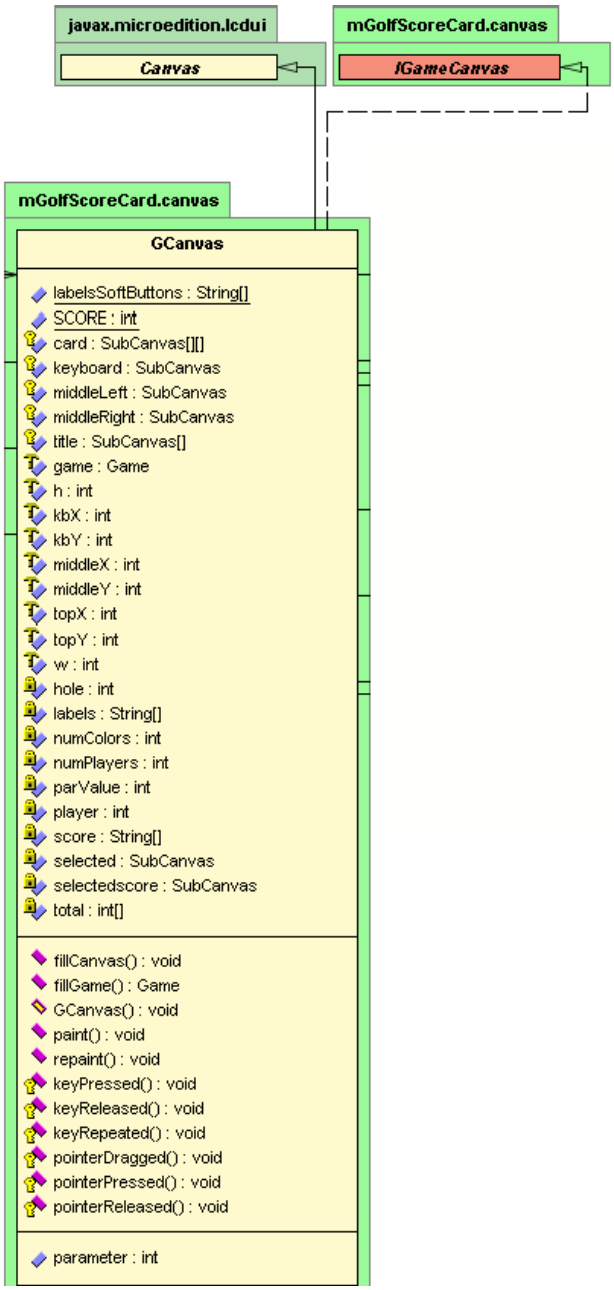


Figura 6-21 Estrutura da classe GCanvas

Classe que serve para mostrar um ecrã de introdução dos dados e garantir que estes sejam guardados, pois implementa a interface `IGameCanvas` que foi implementada de maneira a garantir que a informação do jogo que é preenchida neste ecrã seja guardada.

Com este ecrã é possível introduzir o número de tacadas que os jogadores dão em cada buraco. É possível ver o número de cada jogador, o par do buraco, o número do buraco e a pontuação que o jogador teve nesse buraco.

Este menu tem algumas funcionalidades específicas. Mostrar tanta informação não é algo fácil. A organização da informação pretende mostrar o máximo de informação útil no menor espaço possível.

A navegação no menu é simples, bastando movimentar as setas para cima ou para baixo, ou então, pressionar a zona onde se quer introduzir os dados. Como não existe um cursor, a zona onde deve introduzir os dados tem um parêntesis envolvendo o número do jogador actual, o que facilita ao utilizador saber onde está. Para mais fácil compreensão veja-se a Figura 6-22.

Figura 6-22 Visualização do efeito provocado pela classe GCanvas

Classe SCFCanvas

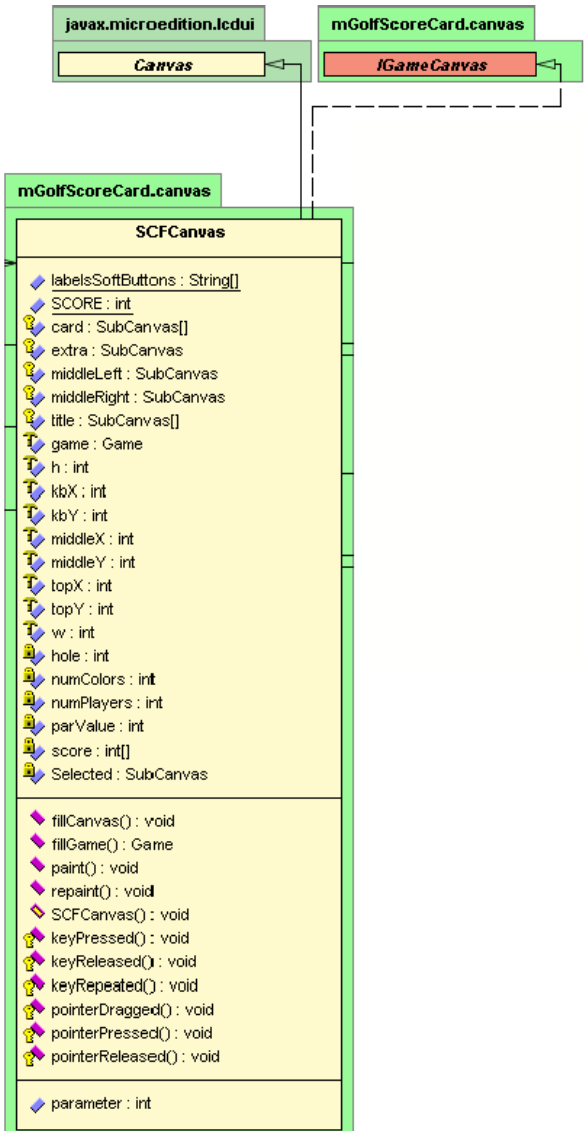


Figura 6-23 Estrutura da classe SCFCanvas

Classe que serve para mostrar o ecrã final com os resultados de cada um dos participantes.

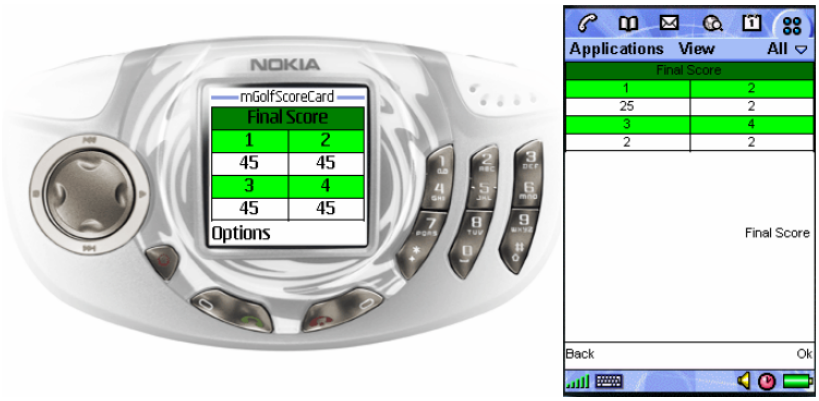


Figura 6-24 Visualização do efeito provocado pela classe SCFCanvas

6.5 *Protocolo de envio de mensagens*

É nesta secção que se encontra descrita, talvez, a parte mais crítica de todo este trabalho. A resposta à pergunta "Como incluir na aplicação a tal conectividade *peer-to-peer* tão desejada?"

Aparentemente, e após o seu estudo, o JXME demonstrava ser a escolha acertada. Infelizmente, após um período de largas horas de experimentação, descobriu-se que o JXTA, e em particular o JXME, não suporta *firewalls*. Ora, como no local onde a aplicação foi desenvolvida existe uma *firewall*, e não havendo alternativas viáveis, optou-se por uma solução própria fazendo o desenvolvimento de um protocolo para troca de mensagens.

Como foi estudado no capítulo 3, a ligação entre dispositivos móveis usando J2ME apenas é possível recorrendo ao um servidor de apoio (*relay server*) que faça a ponte entre os dispositivos (Arquitectura *Middleman*). Concretamente o *relay* vai indicar quais os campos de golfe partilhados e por quem. Quando um utilizador pretende fazer a descarga de um campo partilhado, esta é feita, de facto, a partir do dispositivo original e não do servidor, tratando-se verdadeiramente de um aplicação *peer-to-peer*.

Assim, de seguida, será feita uma abordagem a algumas tecnologias que entraram na implementação do protocolo de envio de mensagens.

6.5.1 *Ligação à rede de um dispositivo MIDP*

Os programadores podem desenvolver aplicações cliente com acesso à rede, clientes que se podem ligar a serviços sem fios usando os protocolos de rede *standard*. A especificação

MIDP requer que os dispositivos suportem HTTP, o mesmo protocolo que os *web browsers* usam.

Este último ponto é particularmente importante, porque apenas é necessário perceber HTTP e as APIs para redes com HTTP, mesmo que a tecnologia a dominar seja consideravelmente mais complexa. Dependendo da rede de transporte usada, um pedido HTTP feito por um dispositivo MIDP pode passar por vários protocolos que podem ou não ser baseados em TCP/IP. Por último, a rede de transporte deve garantir que o destinatário recebe o pedido HTTP intacto, assim como que a resposta HTTP gere uma resposta válida para o dispositivo MIDP.

6.5.2 Java Servlet

A tecnologia Java Servlet [Servlet] possibilita aos programadores *Web* um mecanismo simples e consistente para estender a funcionalidade de um servidor *Web*. Uma *servlet* é uma espécie de *applet* mas do lado do servidor e sem uma cara visível. As *servlets* possibilitam a criação de muitas aplicações *Web*.

As *Servlets* são a escolha da plataforma tecnológica Java para estender e melhorar os servidores *Web*. Baseada em componentes, independente da plataforma e que permite criar aplicações baseadas em *Web*, sem as limitações de performance dos programas CGI que a cada pedido criam um novo processo, entupindo o sistema. Ao contrário dos mecanismos de extensão proprietários existentes, as *servlets* são independentes da plataforma. O que nos deixa livres para escolher o servidor mais adequado, assim como a plataforma, ferramentas, etc.

As *servlets* têm acesso à família inteira das APIs Java, inclusive a API JDBC para aceder a bases de dados. Têm, também, acesso à biblioteca específica `http` e assim têm todos os benefícios da Linguagem Java: portabilidade, performance, reutilização e protecção contra falhas.

6.5.3 Apache TomCat Jakarta

Tomcat [TomCat] é um contentor para *servlet* que é usado como uma referência oficial para as tecnologias *Java Servlet* e *JavaServer Pages*. Para testar a *Servlet* criada foi instalado este contentor, que de uma maneira simples permite testar a *Servlet* localmente.

6.5.4 Comunicação Cliente-Servidor entre MIDlets e Servlets [J2METips]

Muitas das aplicações MIDP utilizam funcionalidades do lado do servidor. O paradigma cliente/servidor, é usado para retirar o trabalho complicado dos dispositivos limitados para o ambiente servidor. A especificação MIDP apenas suporta HTTP.

Toda a comunicação cliente/servidor deve ser feita através de um *gateway* HTTP. As MIDlets podem comunicar com *servlets* ou *Java Server Pages* (JSP) que estejam a funcionar num servidor *web*. As técnicas usadas também servem para comunicar com outras aplicações servidor escritas noutras linguagens como o Perl. Mas usar a mesma linguagem em ambos os extremos torna o código mais simples.

A conectividade HTTP é uma das exigências do perfil MIDP. Um dispositivo com capacidades MIDP deve ser capaz de interagir com um servidor através dos pedidos convencionais. Se a rede não suportar directamente HTTP, então o dispositivo deve transportar esses pedidos através de um *gateway*, mas isso é algo que deve ser transparente.

Em MIDP, podemos interagir com a rede através da estrutura genérica CLDC. A interface `HttpConnection` torna simples interagir com qualquer *site* na *Internet* .

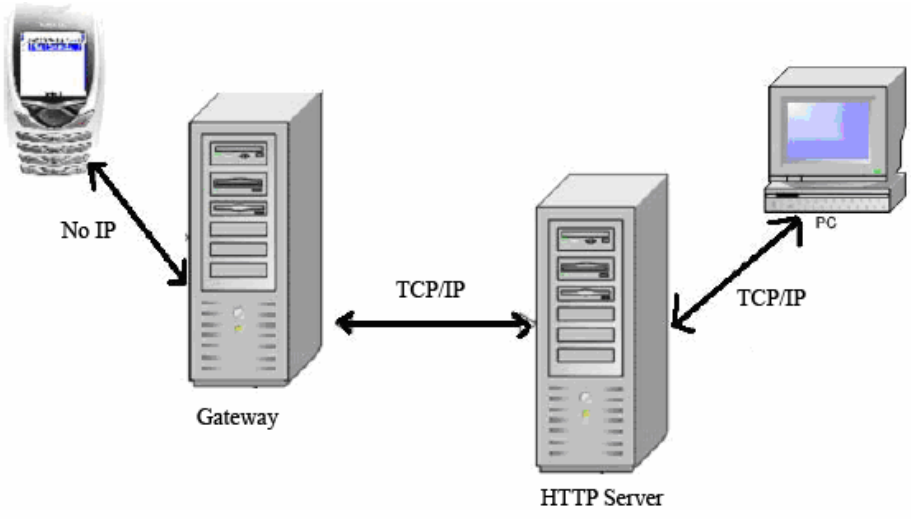


Figura 6-25 Ligação à rede desde um dispositivo MIDP

Além disso, a ligação HTTP faz a ponte entre os dispositivos J2ME e um servidor de aplicações J2EE (*Java 2 Enterprise Edition*). Como ambos suportam HTTP podem, assim, facilmente usar o protocolo para comunicar um com o outro.

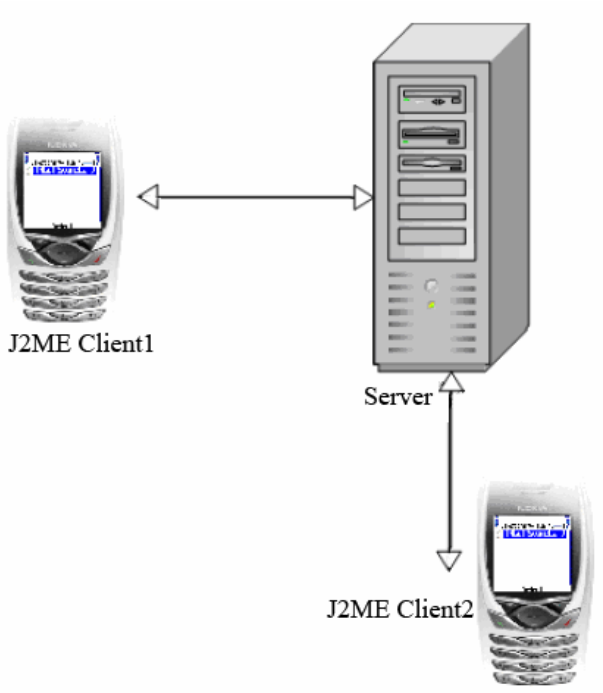


Figura 6-26 Ligação entre dispositivos J2ME e J2EE

A arquitectura de uma aplicação J2EE que sirva clientes sem fios é semelhante à que serve clientes com fios:

Uma aplicação cliente que seja implementada usando MIDP, ou seja, uma MIDlet cliente, fornece a interface com o utilizador para um dispositivo móvel. A MIDlet comunica com uma *Servlet* Java, normalmente via HTTP, e sobre um canal seguro quando necessário.

A *Servlet* interpreta os pedidos da MIDlet e, em troca, satisfaz os pedidos do cliente para componentes EJB (*Enterprise JavaBeans*). Quando os pedidos forem satisfeitos, a *Servlet* gera uma resposta para a MIDlet.

Os componentes EJB encapsulam a lógica da aplicação. Um contentor EJB dispõe de serviços *standard* como transacções, segurança e gestão de recursos, para que o programador se concentre na lógica de negócio.

A *Servlet* e os componentes EJB podem usar APIs adicionais como, por exemplo, JDBC para acesso a base de dados e `JavaMail` para enviar mensagens de correio electrónico.

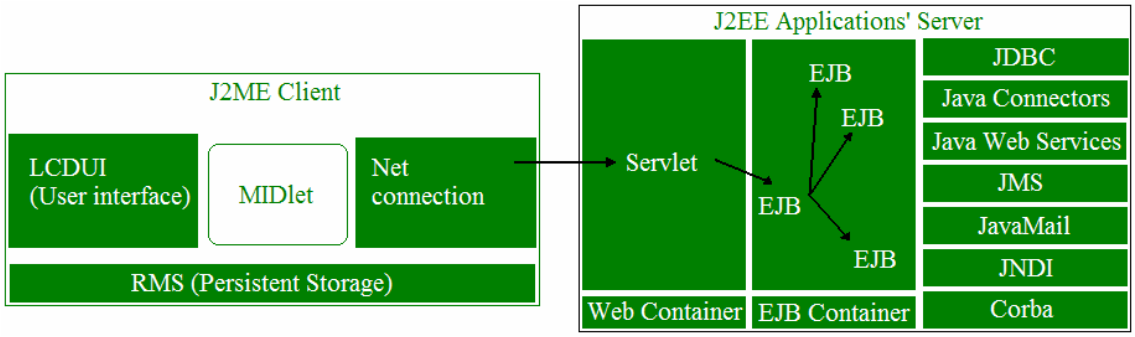


Figura 6-27 Arquitectura de alto nível de uma aplicação J2EE

6.5.5 *Suporte para vários tipos de clientes*

Os programadores para a plataforma J2EE devem ter em conta que esta plataforma incentiva o uso de componentes reutilizáveis através do uso de *enterprise beans*. As aplicações devem ser feitas de forma a que os seus componentes suportem múltiplos tipos de clientes com o mínimo de impacto na lógica de negócio da aplicação.

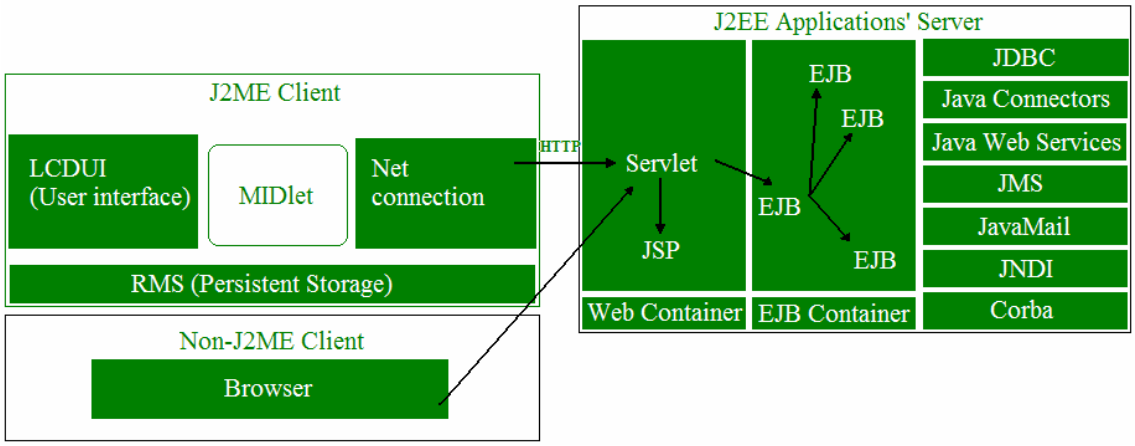


Figura 6-28 Arquitectura de alto nível de uma aplicação J2EE que suporte um cliente J2ME e um *browser*

6.5.6 Considerações na implementação e desenho

Existem várias considerações a ter em conta quando se desenham e se implementam aplicações sem fios do tipo J2EE. Existem muitas limitações quando se criam este tipo de aplicações:

Limitações no desenho

As limitações dos dispositivos móveis impõem muitas restrições no desenho das aplicações sem fios, que são dispositivos muito limitados. As aplicações *wireless* J2EE, em particular, são especialmente restringidas, porque se baseiam na rede. As limitações impostas numa rede móvel são, em geral, maiores do que as de um típico *web browser*, por exemplo:

- Elevado tempo de latência
- Largura de banda limitada
- Conectividade intermitente

Para ultrapassar essas limitações, um cliente MIDP deve seguir estas linhas:

- Apenas se ligar à rede quando precisa
- Consumir apenas os dados da rede que precisa
- Continuar útil mesmo quando desconectado da rede

Troca de Mensagens

O MIDP não inclui mecanismos de comunicação sofisticados cliente/servidor, como *Java Remote Method Invocation* (RMI) ou a API Java para XML baseada em *Remote Procedure Calls* (JAX-RPC). Assim, deve-se usar um protocolo de troca de mensagens escolhendo e programando um determinado formato.

Para a maioria das aplicações, o HTTP funciona perfeitamente como a base de um protocolo de mensagens. Como todos os dispositivos MIDP suportam HTTP a aplicação poderá ser portável através dos vários dispositivos.

O HTTP é *firewall-friendly*. A maioria dos servidores está separada dos clientes móveis por *firewalls* e o HTTP é um dos poucos protocolos que passa através de *firewalls*.

Formato da Mensagem

Assim, foi escolhido um formato para as mensagens do protótipo que se divide em quatro partes:

1. Tipo da Mensagem
2. Nome de quem envia a mensagem
3. Nome para quem se envia a mensagem
4. Dados enviados na mensagem

Tipo da Mensagem

Porque não utilizar o XML no formato da mensagem? A principal razão é que um *parser* XML ocupa muito espaço no ficheiro `jar` da aplicação. Outra razão, é o aumento do tamanho da informação enviada, o que acaba por ser desnecessário. No entanto, futuramente não fica excluído o XML, pois o Servidor pode transformar os dados para XML uma vez que não tem as limitações dos dispositivos móveis.

O protocolo implementado prevê cinco tipos de mensagens:

- **LOGIN:** quando o utilizador quer partilhar um campo, envia o nome do campo como dado da mensagem
- **LOGOUT:** quando o utilizador quer deixar de partilhar o campo que estava a partilhar
- **REQUEST_COURSE:** quando o utilizador pretende que lhe enviem um campo, envia o nome do campo como dado
- **SEND_COURSE:** quando o utilizador envia um campo que lhe foi requisitado por outro utilizador, o nome do destinatário é o utilizador que vai receber esse campo

- **SEND_MAIL:** quando o utilizador envia um jogo destinado ao *email* que foi especificado no campo do destinatário da mensagem

6.5.7 Modelação das classes na parte da Servlet

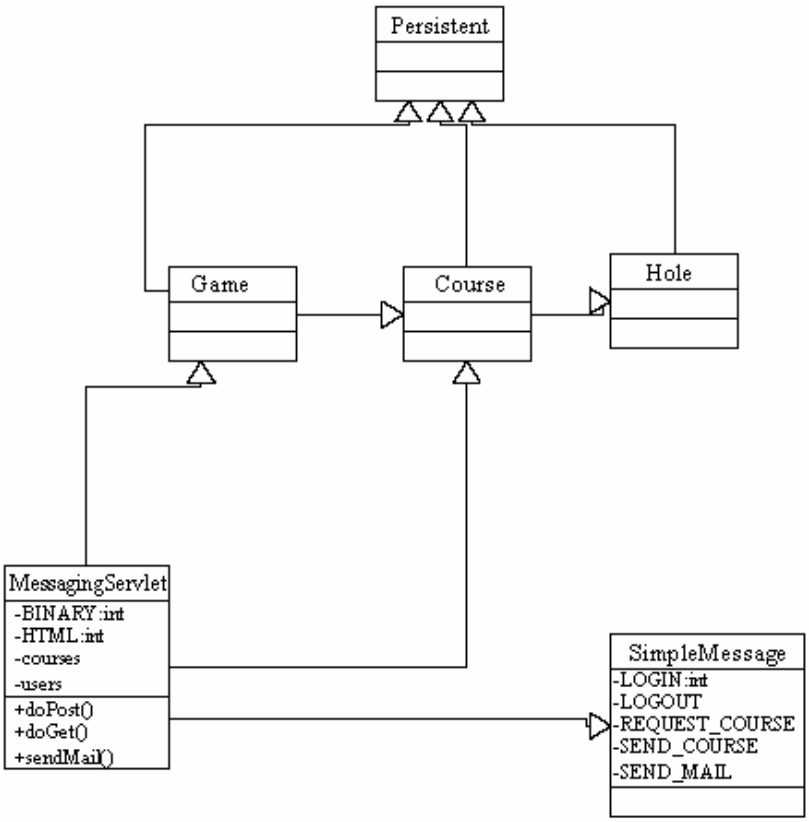


Figura 6-29 Hierarquia de classes da *servlet*

Classe MessagingServlet

Classe principal, trata os pedidos feitos pelos utilizadores, encarrega-se do envio das mensagens aos seus destinatários e, além disso, permite que tanto os utilizadores dos dispositivos móveis como os dos *web browsers* normais possam saber exactamente quais os campos partilhados.

No método GET, a *Servlet* analisa se o pedido vem de um dispositivo MIDP ou de um *web browser* normal, uma vez que os dados enviados são diferentes.

Enquanto que, para a MIDlet são enviados os nomes dos campos de forma a que possam aparecer numa lista ou receber o tratamento que se entender com eles, para o *web browser* será gerada uma tabela HTML para amostragem da informação, como se pode ver na Figura 6-30.

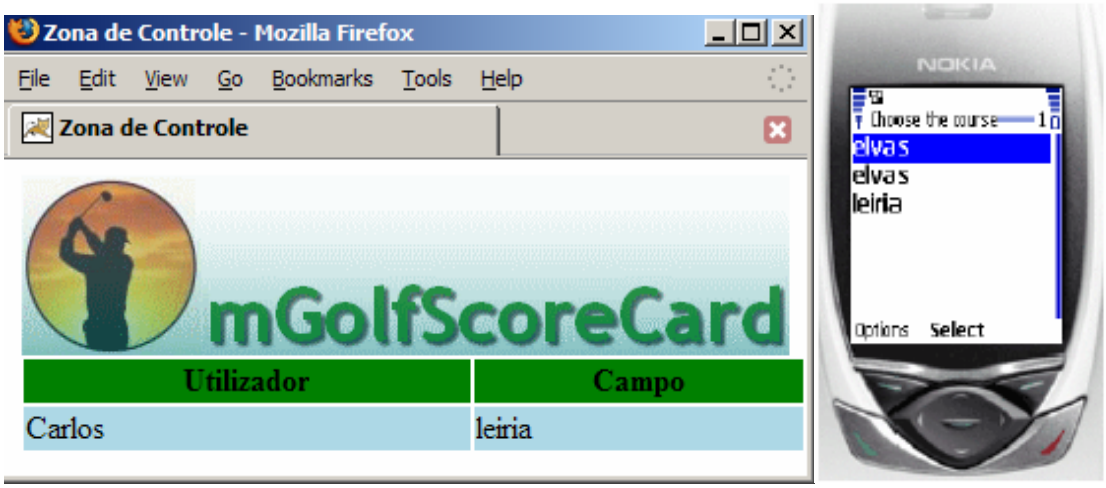


Figura 6-30 Aspecto num *web browser* e num dispositivo J2ME já tratado

6.5.8 *Modelação das classes na parte do cliente MIDP*

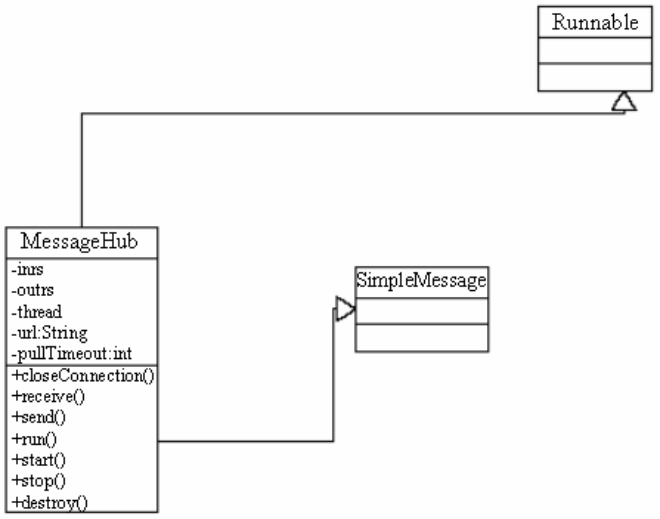


Figura 6-31 Modelo de classes do cliente J2ME

Desta forma, foi criado um mecanismo simples de armazenamento e envio de mensagens.

A parte mais importante num mecanismo deste tipo é a que controla o tráfego de mensagens, um *Hub* de Mensagens. O Hub é a classe usada pela aplicação para enviar e receber mensagens. Usa dois *record stores*: um para registar as mensagens que chegam e outro para guardar as mensagens a enviar. As mensagens são recebidas e enviadas por uma *thread* que corre em *background*. O modo de usar o Hub é bastante simples pois apenas necessita de dois parâmetros no seu construtor: o URL da *servlet* e um valor do *timeout*.

Como foi dito anteriormente, estão associados dois *record stores* ao *Hub*.

Estes possibilitam um armazenamento de persistência necessário para construir um sistema de mensagens de confiança.

O verdadeiro trabalho é feito por uma *thread* em segundo plano que corre no método run(). Quando uma mensagem é adicionada ao *record store* ou quando atinge o *timeout*, a *thread* "acorda" e manda um pedido POST para a *servlet*, enviando uma mensagem, se for pretendido, com o corpo do pedido. A *servlet* responde com uma mensagem de resposta. Posteriormente, a *thread* adiciona, então, a mensagem para o *record store* das mensagens recebidas e "acorda" a *thread* que estava à espera.

6.5.9 Explicação da utilização do protocolo

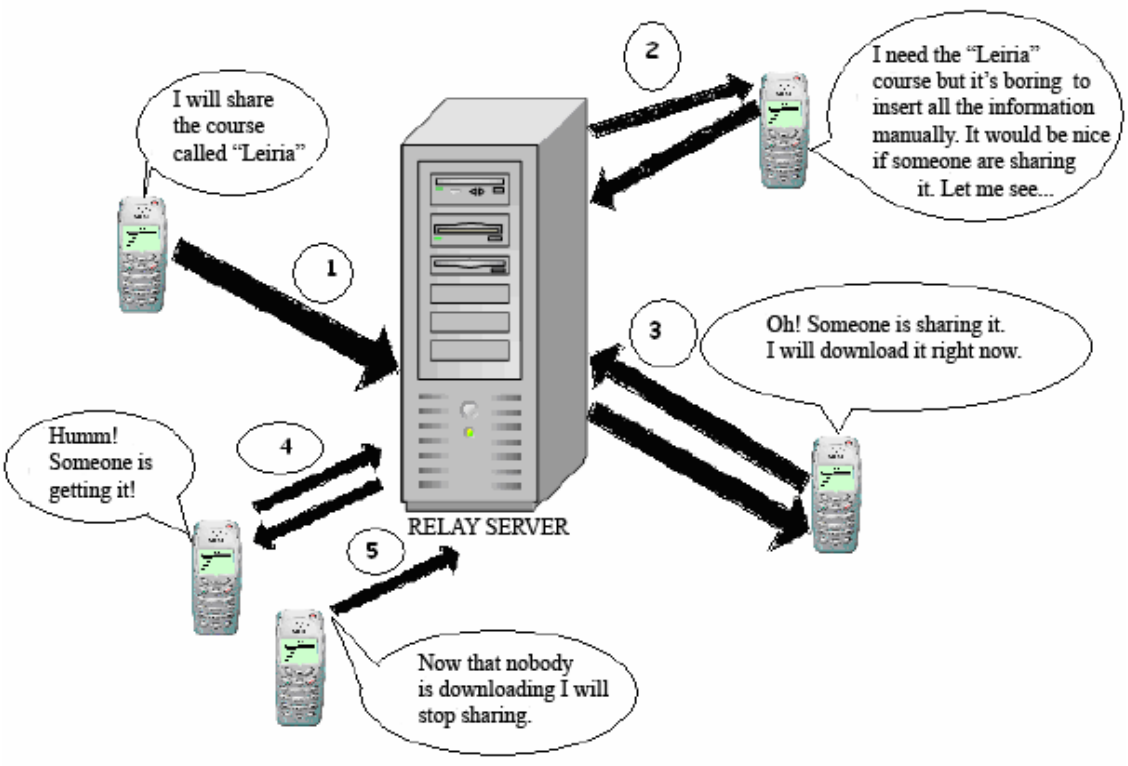


Figura 6-32 Exemplificação da utilização do protocolo usado

1- Envio da mensagem de LOGIN para iniciar a partilha de campos.

2- Envio do método GET para obter a lista de campos partilhados.

3- Envio da mensagem REQUEST_COURSE para pedir um campo e aguardar que o servidor lhe envie um campo depois de o pedir ao outro utilizador que o tem partilhado.

4- Recebe a mensagem REQUEST_COURSE e envia a mensagem SEND_COURSE com o campo pedido.

5- Envia a mensagem LOGOUT para que o servidor o retire dos campos partilhados.

Na Figura 6-33 pode-se ver a Arquitectura Geral do Protótipo: o Peer1 com o mGolfScoreCard partilhando informação de um campo de golfe com o Peer2. O Peer3 (WebBrowser) visualiza a informação de todas as partilhas a decorrer.

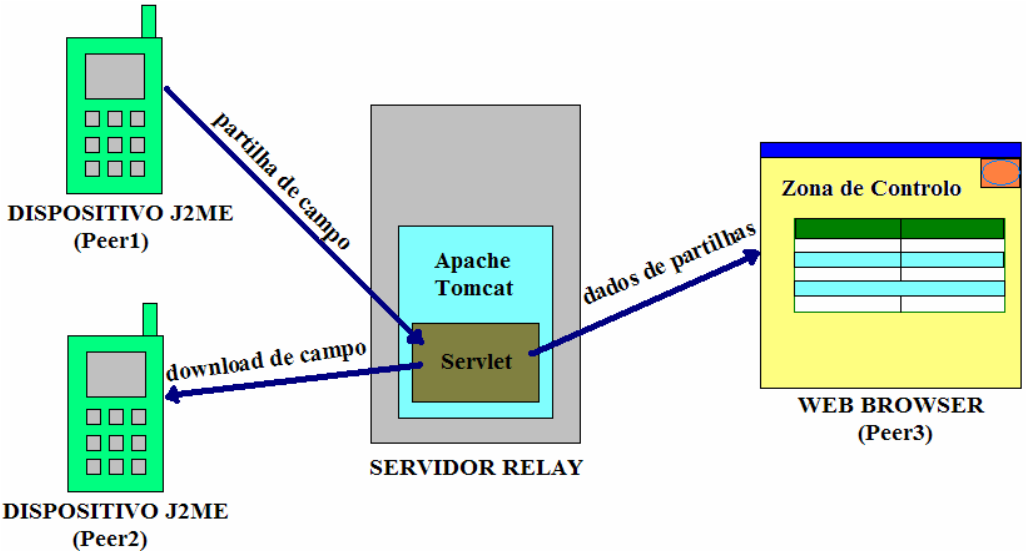


Figura 6-33 Arquitectura geral do protótipo.

6.6 *Estrutura do código da MIDlet e documentação*

O código está documentado de acordo com as *Java Code Conventions*.

A estrutura dos *packages* é a que se pode ver na Figura 6-34.

A aplicação, assim como a documentação, foi implementada em inglês com o propósito de algum dia poder vir a ser alterada mais facilmente por alguém.

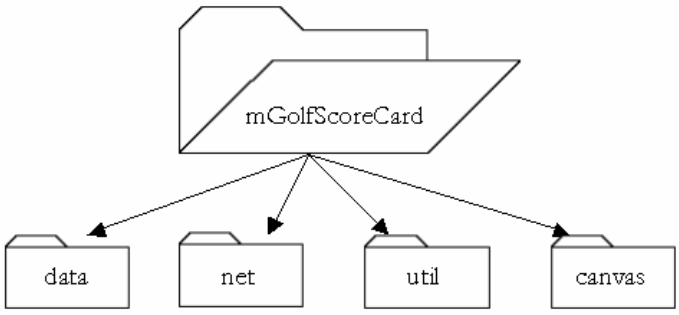


Figura 6-34 Estrutura dos *packages*

- **data:** contém as classes que guardam informação útil para a aplicação
- **net:** contém as classes úteis para a parte de *peer-to-peer*
- **util:** contém as classes auxiliares
- **canvas:** contém as classes para a parte da interface

6.7 Modelo de classes para guardar informação

A principal função da aplicação será guardar a informação necessária para depois calcular os resultados no campo de golfe. Com esse intuito, desenvolveu-se este modelo de classes que é suficiente para guardar todos os dados necessários, sendo de realçar que não existe persistência em J2ME, ao contrário das versões para outras plataformas, tendo que ser, portanto, implementada de raiz (interface `Persistent`).

De seguida, são descritas e explicadas as classes implementadas:

Classe Player

Representa a entidade Jogador e guarda todos os atributos importantes relativos a um jogador de golfe. Implementa, também, a interface para permitir persistência.

Classe Hole

Representa a entidade Buraco guardando todos os atributos importantes relativos a um buraco de um campo de golfe. Nesta classe existem informações essenciais para se poder efectuar um jogo e implementa também a interface para permitir persistência.

Classe Course

Guarda todos os atributos importantes relativos a um campo de golfe, que pode ter 9 ou 18 buracos. Nesta classe os atributos mais importantes são o número de buracos e a respectiva lista, de maneira a guardar essa informação essencial para poder efectuar jogos.

Classe Game

Esta classe guarda todos os atributos importantes relativos a um jogo de golfe. Existem vários atributos importantes nesta classe, como o número de jogadores, o par de cada buraco, o número de tacadas efectuadas pelos jogadores em cada buraco, o resultado final do jogo, etc.

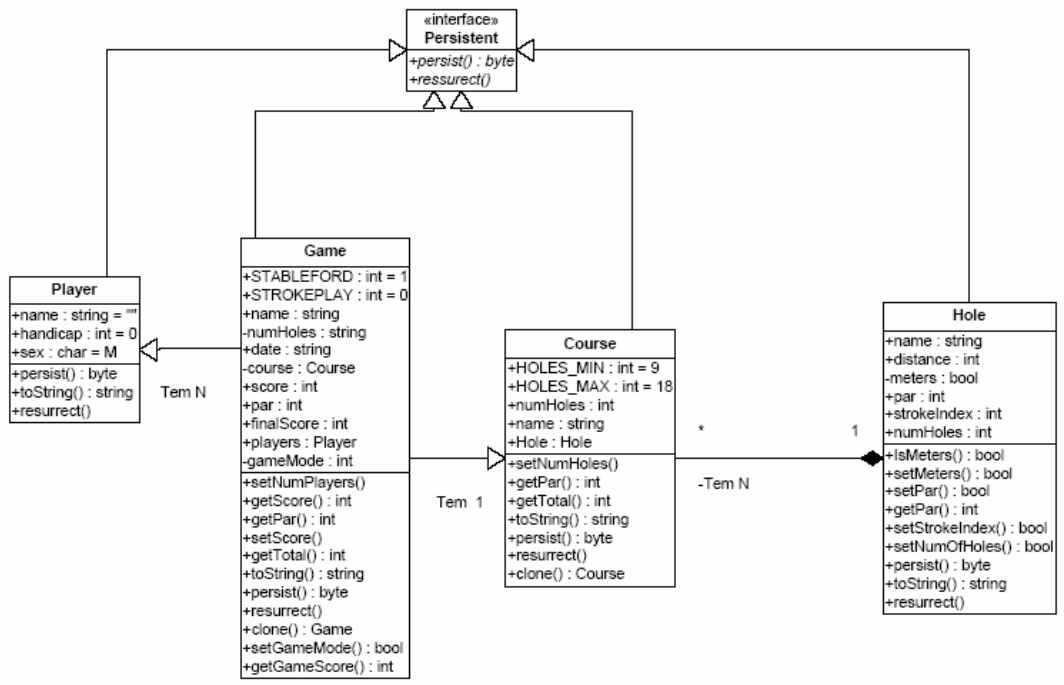


Figura 6-35 Hierarquia de classes que serve para guardar a informação necessária na aplicação

6.8 Considerações tidas em conta para minorar as limitações dos dispositivos

Para um melhor aproveitamento da memória dos telemóveis e da sua capacidade de processamento foram tidos em conta vários aspectos:

- minimização do uso de encapsulamento
- libertação de recursos à medida que deixam de ser necessários
- sempre que possível, o nível de acesso usado para os atributos foi privado, uma vez que, ofuscada a aplicação, reduz o tamanho do ficheiro JAR ao mínimo.

6.9 Testes realizados

6.9.1 Funcionalidade

Optou-se pela metodologia de desenvolvimento *Extreme Programming* (XP) [XP] devido à rapidez que imprime no desenvolvimento e teste de aplicações. Uma vez que os casos de uso do mGolfScoreCard são muito concretos e específicos, esta metodologia tornou-se a mais indicada. Assim, foram elaborados testes unitários a cada uma das funcionalidades da aplicação:

Inserir um campo:

- inserir o nome do campo (passou a 100%)
- inserir o número de buracos (passou a 100%)
- para cada buraco inserir: nome, par, *stroke index* e distância (passou a 100%)

Inserir um jogo:

- escolher um campo (passou a 100%)
- escolher um modo de jogo (passou a 100%)
- escolher número de jogadores (passou a 100%)
- ocorrências por buraco e por jogador (passou a 100%)

Partilhar um campo:

- mudar configurações (passou a 100%)
- seleccionar campo para partilhar (passou a 100%)
- efectivar a partilha (ocorreram alguns erros mas, aparentemente derivados à intermitência do serviço GPRS da operadora utilizada)

Carregar um campo de outro utilizador

- visualizar os campos que existem para *download* (ocorreram erros mas deveram-se à intermitência do serviço GPRS da operadora utilizada)

- fazer *download* de um campo (ocorreram erros mas deveram-se também à intermitência do serviço GPRS da operadora utilizada)

Saliente-se o facto de se ter utilizado, no desenvolvimento e testes, emuladores de dispositivos móveis mais conhecidos e preferencialmente distintos:

- com pequenos e grandes ecrãs

- de fabricantes diferentes

- com teclado físico *vs* com *touch screen.*

6.9.2 Usabilidade

Realça-se o facto de, nesta fase de testes de usabilidade, ter-se feito uso de dispositivos reais. Os testes de usabilidade tornam-se numa importante ferramenta uma vez que permitem:

- reduzir o tempo que se demora a executar tarefas

- reduzir o número de erros cometidos

- reduzir o tempo de aprendizagem

- aumentar a satisfação das pessoas com o sistema

Foram realizados testes com possíveis utilizadores do mGolfScoreCard. Para isso, foi pedido aos utilizadores para, numa primeira fase, realizarem tarefas concretas; depois que respondessem ao um pequeno questionário e emitissem sugestões para o melhoramento da aplicação. O próximo capítulo aborda mais aprofundadamente os testes de usabilidade realizados e respectivas conclusões.

6.10 Sumário

Neste capítulo foram descritas as metodologias adoptadas no desenvolvimento do protótipo mGolfScoreCard. Salientaram-se pontos que, usualmente, passam despercebidos na programação de aplicações *desktop*. Uma bateria de restrições e limitações são uma realidade nos dispositivos móveis e deve-se saber contornar cada uma delas. Fez-se a descrição das classes implementadas e os truques usados para maximizar a performance e, em simultâneo, diminuir o tamanho final.

Foi também descrito o protocolo implementado para a troca de mensagens entre *peers* (dispositivos móveis) para se proceder à partilha de campos de golfe.

7 Avaliação do protótipo

Neste capítulo, será exposto o estudo de usabilidade realizado para avaliação do protótipo - mGolfScoreCard. Assim, foi pedido a 5 utilizadores que respondessem a um inquérito por forma a poder-se determinar os níveis de usabilidade e utilidade da aplicação. Na próxima secção é descrita a forma como os testes foram elaborados e nas duas secções subsequentes são descritos os resultados obtidos e as conclusões tiradas.

7.1 Participantes e Metodologia

Foi pedido a 5 pessoas, de certa forma familiarizadas com o jogo de golfe, que participassem nos testes a efectuar ao protótipo, respondendo a um inquérito (ver Anexo C), à medida que utilizavam a aplicação e efectuavam algumas tarefas específicas. Esses participantes eram de diversas áreas, estudantes e profissionais, homens e mulheres (ver Figura 7-1).

Participant	Age Range (Gender)	Occupation	Mobile Device Used
A	18-23 (F)	Student	Siemens M50
B	18-23 (M)	Student	Motorola V975
C	24-32 (M)	Teacher	Qtec 9090
D	24-32 (F)	Teacher	Nokia 6600
E	33-40 (M)	Golfer	Motorola Accompli 008

Figura 7-1 Informação demográfica sobre cada participante e o dispositivo móvel usado por cada um

Para além disso, os participantes escolhidos eram também de várias classes etárias, de diversas ocupações e com dispositivos móveis distintos. Este último aspecto tornou-se importante, na medida em que se pôde testar o funcionamento da aplicação em dispositivos:

- a cores e monocromáticos ou com tons de cinzento (participantes **B**, **C** e **D** com dispositivos a cores; **A** com 2 cores e **E** com tons de cinzento)

- com teclado físico, com *touch screen* e com ambos (participantes **A**, **B** e **D** dispositivos com teclado físico; **E** com *touch screen* e **C** com ambos)

Quanto à metodologia usada, foi pedido aos utilizadores que respondessem a um inquérito onde, além dos dados pessoais, estes deveriam realizar um conjunto de tarefas previamente definidas, responder a um questionário com respostas discretas (cujas respostas possíveis eram Mau, Fraco, Suficiente, Bom e Muito Bom) e ainda a algumas perguntas gerais sobre a aplicação.

De notar que alguns dados foram previamente inseridos na aplicação antes dos testes, por forma a tornar possível a resposta a algumas perguntas, por parte dos participantes. Isto é, foram inseridas informações sobre alguns campos e jogos. Foi igualmente distribuído um manual do utilizador da aplicação (ver Anexo D) que os utilizadores poderiam consultar, caso necessitassem.

7.2 Observações

De salientar que todas as tarefas da primeira parte do inquérito foram concluídas com sucesso por cada um dos participantes. Cada uma dessas tarefas consistia num pequeno caso de uso: consultar determinada característica de um campo, consultar o resultado de um jogo, alterar o nome de um jogador, inserir um campo, partilhar um campo, etc.

Relativamente ao questionário apresentado a seguir às tarefas, os resultados obtidos encontram-se resumidos na Figura 7-2:

	A	B	C	D	E	Média
Aspecto Gráfico/Visual:						
Menus	5	4	5	5	5	4,8
Cores	4	4	4	5	5	4,4
Qualidade gráfica	4	3	4	4	5	4
Legibilidade	4	3	5	4	4	4
Funcionalidade:						
Inserção de campos	4	4	4	5	5	4,4
Edição de campos	4	3	4	3	4	3,6
Remoção de campos	4	4	3	4	5	4
Criação de um jogo	4	4	5	3	5	4,2

Edição de um jogo	4	3	4	4	4	3,8
Remoção de um jogo	4	4	3	4	5	4
Partilha de campos	4	3	4	4	5	4
Facilidade geral de utilização	4	4	4	4	5	4,2
Utilidade para o jogo de golfe						
Controlo da classificação do jogo	5	4	5	5	5	4,8
Comunicação entre jogadores	4	3	3	4	4	3,6
Acompanhamento *off-online* do jogo	4	3	4	3	5	3,8
Utilidade geral para o jogo de golfe	5	4	4	5	5	4,6
Apreciação global	4	4	4	4	5	4,2

Figura 7-2 Respostas ao questionário onde 1-Mau, 2-Fraco, 3-Suficiente, 4-Bom e 5-Muito Bom

Outra observação interessante consiste no facto de os utilizadores com dispositivos com ecrã monocromático acharem o aspecto gráfico bastante satisfatório à semelhança dos que dispunham de ecrã a cores (ver Figura 7-3).

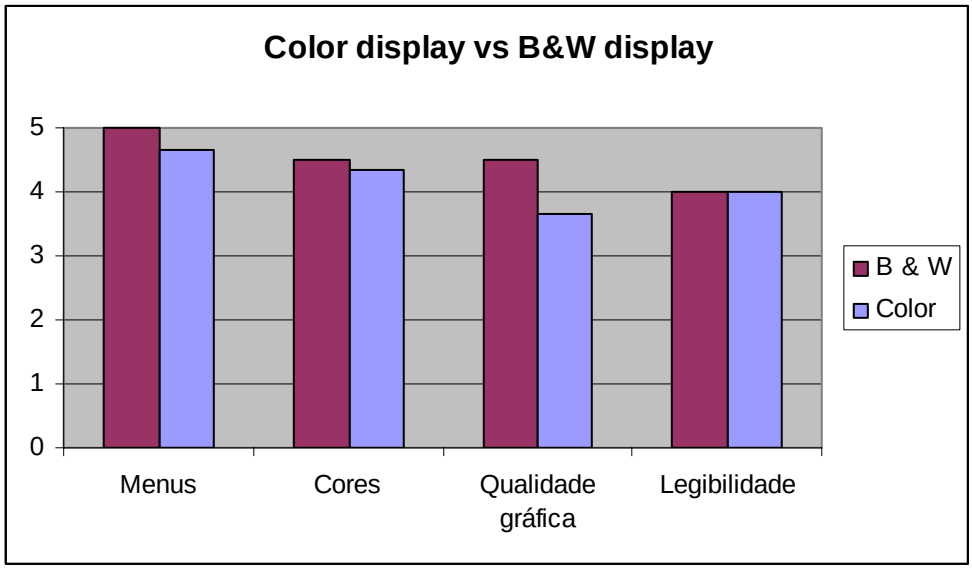


Figura 7-3 Comparação entre as opiniões dos utilizadores que usaram dispositivos a cores e os que usaram dispositivos monocromáticos ou com tons de cinzento

Outro aspecto interessante de comparar é o comportamento da aplicação em dispositivos com e sem teclado físico. Mais uma vez a satisfação dos participantes torna-se evidente independentemente do dispositivo móvel utilizado (ver Figura 7-4). No entanto, em relação a este aspecto há algumas considerações pertinentes que serão discutidas na próxima secção.

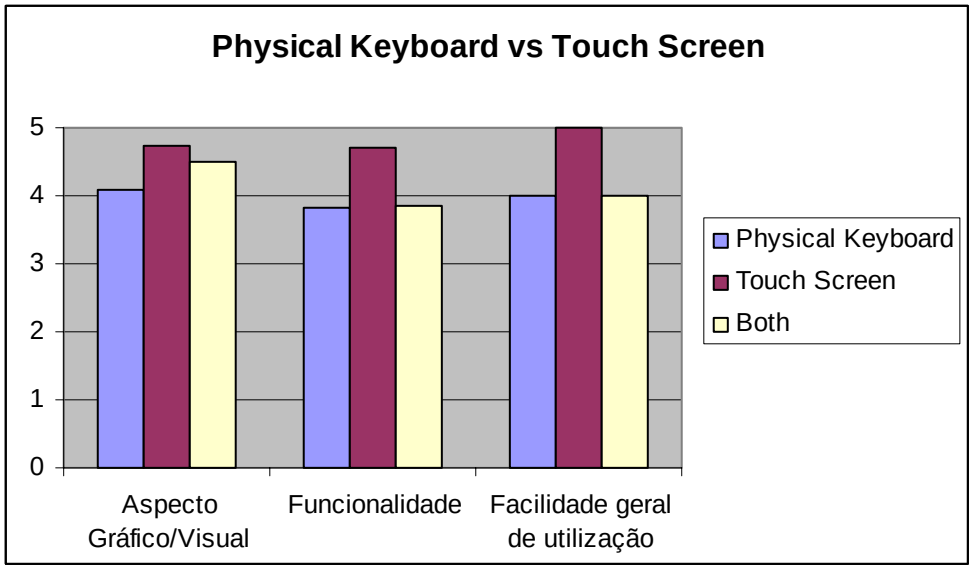


Figura 7-4 Comparação entre as opiniões dos utilizadores que usaram dispositivos com teclado físico, os que usaram dispositivos com *touch screen* e ambos

Com base nos resultados do teste de usabilidade pode-se constatar que a interacção com o utilizador e as funcionalidades da aplicação estão no bom caminho.

7.3 Discussão

Nesta secção são tecidas algumas conclusões sobre o inquérito.

Os participantes, apesar das "boas notas" dadas, fizeram observações e comentários bastante interessantes e pertinentes que devem ser tidos em conta em versões posteriores do mGolfScoreCard. Vejamos:

Dispositivos com teclado físico e *touch screen*

O participante C, possuidor de um Qtec 9090 com teclado físico e *touch screen*, detectou que a aplicação não respondia aos eventos do teclado. Isto deve-se ao facto de o protótipo considerar que quando um dispositivo tem eventos de rato, à partida, não terá teclado

físico. Obviamente, esta regra para alguns dispositivos actuais não faz sentido sendo necessário corrigir este problema no futuro.

Consulta de dados mais eficiente

Outra das observações interessantes prende-se com a consulta de dados na aplicação. Normalmente, quando se fala em inserção de dados, pensa-se também na remoção, alteração e consulta. Ora, neste protótipo houve a junção da alteração e da consulta dos dados. É claro que não terá sido a melhor escolha uma vez que não faz sentido consultar informação editando-a. Por exemplo, para consultar o resultado final de um jogo é necessário percorrer (editar) toda a informação de cada buraco do campo onde foi efectuado o jogo. Mais um tópico a rever em futuras implementações.

Partilha de mais informação

No protótipo apenas é possível partilhar informação sobre campos de golfe. Seria, no entanto, interessante que houvesse também a possibilidade de partilhar informação de jogos e até mesmo de jogadores.

Ainda no âmbito da partilha, seria interessante usar outros protocolos como o *Bluetooth* e o IrDA e mais ainda, ser a própria aplicação a detectar qual dos protocolos a usar em cada situação.

Muito embora uma enorme evolução nas capacidades de conectividade, armazenamento, processamento, memória, multimédia, etc. destes dispositivos seja ainda esperada e também que o J2ME evolua paralelamente, apesar destas sugestões por parte dos participantes, o estudo efectuado comprova inequivocamente que a partilha de dados entre dispositivos móveis numa rede *peer-to-peer* é possível. Na minha opinião, será mesmo o futuro, dada a característica ímpar deste tipo de dispositivos que oferecem disponibilidade permanente independentemente da localização.

Se observarmos com atenção o que a Java Community Process está a preparar, facilmente deduzimos que os dispositivos móveis terão um futuro bastante promissor. Enumeram-se abaixo algumas JSRs (*Java Specification Requests*) [JSR] como exemplo:

- JSR 120: Wireless Messaging API

- JSR 135: Mobile Media API
- JSR 172: J2ME™ Web Services Specification
- JSR 177: Security and Trust Services API for J2ME™
- JSR 179: Location API for J2ME™
- JSR 184: Mobile 3D Graphics API for J2ME™
- JSR 211: Content Handler API
- JSR 238: Mobile Internationalization API
- JSR 259: Ad Hoc Networking API
- JSR 271: Mobile Information Device Profile 3

Em particular esta última, que adicionará, entre outras, as seguintes funcionalidades interessantes ao MIDP2 actual:

- Permite múltiplas MIDlets concorrentes numa só VM
- Permite MIDlets em *background* (sem Interface Gráfica)
- Permite MIDlets executadas em arranque (por exemplo, iniciadas em *boot time*)
- Permite comunicação entre MIDlets
- Permite bibliotecas partilhadas entre MIDlets

Com estes exemplos, apenas a imaginação e criatividade do programador são o limite.

7.4 Sumário

Ficou descrito como se processou a fase de testes de usabilidade do protótipo, as conclusões possíveis, potencialidades e limitações. Os testes de usabilidade permitiram ainda concluir sobre as reais capacidades de partilha entre dispositivos móveis heterogéneos numa rede *peer-to-peer*. Mostrou-se que a *Java Community Process* está

empenhada em fazer do J2ME o futuro, no que respeita à programação para dispositivos integrados móveis.

8 Conclusão e Desenvolvimentos Futuros

8.1 Conclusão

Os dispositivos móveis têm alcançado enorme sucesso ao longo dos últimos anos e o seu verdadeiro valor começa agora a ser mais notório à medida que vão ganhando capacidades de rede. Mas estes dispositivos têm bastantes limitações e elevada mobilidade sendo necessário que os ambientes de rede saibam lidar com isso. A arquitectura de rede mais apropriada é a distribuída que segue o paradigma *peer-to-peer*, uma vez que estas arquitecturas conseguem lidar com clientes que "entram" e "saem" sem aviso. As arquitecturas *peer-to-peer* preocupam-se com a independência da infra-estrutura da camada da aplicação e encarregam-se desde a descoberta de recursos ao uso desses recursos sem o conhecimento prévio da sua localização.

As soluções de rede que podem existir dependem dos ambientes de programação dos dispositivos. A especificação J2ME, que é o ambiente de desenvolvimento Java para dispositivos móveis, ainda só dispõe de um simples protocolo, o que limita muito as capacidades de rede. Mas o uso da arquitectura *Middleman*, e do seu *relay*, permite aos dispositivos com capacidades Java, juntar-se em aplicações de rede.

Uma das conclusões que se pode tirar do desenvolvimento do protótipo mGolfScoreCard é de que a altura certa para o desenvolvimento destas aplicações ainda está para chegar. A tecnologia J2ME evoluiu para uma nova versão que não traz grandes novidades nas conexões de longa distância e os telemóveis ainda necessitam de maior capacidade para poder comportar aplicações no seu pleno potencial.

Grande parte do tempo foi dedicado à investigação, pois qualquer das tecnologias usadas é recente, facto que me incitou a escrever este relatório de maneira que possa servir também de ajuda para quem no futuro queira desenvolver usando estas tecnologias.

A aplicação desenvolvida dedicou especial atenção à portabilidade. É algo invisível para o utilizador normal mas de grande complexidade para o programador. Como não se espera que os telemóveis venham a ter todos o mesmo formato, será que o Java irá repensar o título? Porque para uma melhor portabilidade faltam rotinas genéricas para tratamento de

input (com teclado físico e/ou com *touch screen*), classes que permitam maximizar a área de visualização disponível em cada dispositivo, o uso de cores, interacção com o utilizador, multimédia, etc.

O projecto JXTA para J2ME (JXME) ainda está também em plena e constante evolução e, ao ser desenvolvido por uma comunidade, tem falhas em termos de disponibilidade de informação e nas aplicações que demonstram a plataforma. Poderá, no entanto, vir a ser, a médio prazo, uma base de sucesso nesta área, dada a sua independência tanto na linguagem de programação como na plataforma.

A funcionalidade *peer-to-peer* acabou por ser pensada de raiz através da criação de um protocolo de mensagens próprio, algo que não estava nos objectivos iniciais. O facto de o JXME não funcionar atrás de *firewalls* fez com que fosse colocado de parte embora a versão mais recente já tenha essa limitação ultrapassada.

De qualquer forma, ficou demonstrada, apesar das muitas limitações existentes, que a comunicação entre diferentes dispositivos móveis numa rede *peer-to-peer* é possível e, na minha opinião, será o futuro. Não tenho dúvidas que, com o aumento das capacidades de processamento, de memória, de armazenamento e de conectividade, farão com que os dispositivos móveis se intercomuniquem e sejam dispositivos com uma utilidade provavelmente impensável há poucos anos atrás.

8.2 Desenvolvimentos Futuros

Existem vários aspectos a melhorar na aplicação protótipo. De seguida é apresentada uma lista de possíveis desenvolvimentos/melhoramentos muitos deles sugeridos pelos utilizadores nos testes de usabilidade:

- Mudança do protocolo desenvolvido para o protocolo JXME, uma vez que o protocolo usado foi criado de raiz não salvaguardando muitos aspectos como a segurança, a tolerância a falhas, etc;

- Adição de mais protocolos de transporte, não só GPRS mas também, *Bluetooth* e infravermelhos, para pequenas distâncias, uma vez que o seu uso é a custo zero;

- Possibilidade de partilhar mais do que um campo em simultâneo;

- Considerar que existem dispositivos móveis recentes que dispõem simultaneamente de teclado físico e ecrã *touch screen*. Por isso, é essencial que a aplicação aceite, nestes casos, *input* de qualquer das fontes. Neste momento, num dispositivo com estas características, a aplicação considera apenas o *input* pelo *touch screen*;

- Elaboração de um componente de menus e comandos associados;

- Actualmente o J2ME tem um sistema de verificação das teclas pressionadas muito difícil de gerir e de difícil extensão da aplicação no futuro. Além disso, a clareza na leitura do código sai bastante prejudicada;

- Elaboração da internacionalização da aplicação. Quando esta conclusão foi escrita já muitos dispositivos tinham abolido as limitações de espaço;

- Realização de uma aplicação para PC de modo a dar apoio a esta aplicação móvel, permitindo a passagem de informação para telemóvel e vice-versa, e gerindo a informação recebida do telemóvel;

- Realização de testes à aplicação mais precisos e aprofundados

9 Referências Bibliográficas

9.1 Livros, revistas e artigos

[Ali2004] Ali Raza Butt, Troy A. Johnson, Yili Zheng, Y. Charlie Hu; Kosha: A Peer-to-Peer Enhancement for the Network File System; November 2004; ISBN:0-7695-2153-3

[Bræk2002] Bræk, Rolv; Husa, Knut Eilif; Melby, Geir; ServiceFrame Whitepaper (draft); Ericsson, May 2002.

[Comp2001] Gong, Li; JXTA: A Network Programming Environment; IEEE Internet Computing, May-June 2001.

[Comp2001] Gong, Li; Sun Microsystems Inc.; JXTA: A Network Programming Environment; IEEE Internet Computing, volume 5, Issue 3, May-June 2001, page 88-95

[Computer] Computer; Volum 34, Issue 7, July 2001, page 21

[Computing] Golden G. Richard III; Service Advertisement and Discovery: Enabling Universal Device Cooperation ; IEEE Internet Computing, September-October 2000.

[Frodigh2000] Frodigh, Magnus; Johansson, Per; Larsson, Peter; Wireless ad hoc networking - The art of networking without a network ; Ericsson Review No. 4 2000 http://www.ericsson.com/about/publications/review/2000_04/article124.shtml

[Gollmann1999] Gollmann, Dieter; Computer Security; John Wiley & sons, 1999

[JXMEwp] Arora, Akhil; Haywood, Carl; Pabla, Kuldip Singh; JXTA for J2ME - Extending the Reach of Wireless With JXTA Technology; Sun Microsystems, Inc; March 2002 http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf

[JXTAvn] Traversat, Bernard; Abdelaziz, Mohamed; Duigou, Mike; Hugly, Jean-Christophe; Pouyoul, Eric; Yeager, Bill; Project JXTA Virtual Network; Sun

Microsystems, Inc. ; February 2002
http://www.jxta.org/project/www/docs/JXTAprotocols.pdf

[Kiely2001] Kiely, Don; Wanted: Programmers for Handheld Devices; Computer, Volum 34, Issue 5, May 2001, page 12-14

[Knudsen2001] Knudsen, Jonathan; Wireless Java: Developing with Java 2 Micro Edition; June 2001

[Kumaran2002] Kumaran, S. Ilango; Jini Technology. An Overview ; Prentice Hall, Inc. 2002

[Mahmoud2002] Mahmoud, Qusay; Advanced MIDP Networking, Accessing Using Sockets and RMI from MIDP-enabled Devices; January 2002
http://wireless.java.sun.com/midp/articles/socketRMI/

[Skaflestad2001] Skaflestad, Odd Arild; Kaurel, Nina; Peer-to-Peer Networking. Configuration Issues and Distributed Processing; Arcticle written in realtion to subject SIE50AC Autumn 2001

[Stang2001] Stang, Mark; Whinston, Stephen; Enterprise Computing with Jini Technology; IT Pro January-February 2001

[Stephanos2004] Stephanos Androutsellis-Theotokis, Diomidis Spinellis; A survey of peer-to-peer content distribution technologies; ACM Computing Surveys (CSUR) Volume 36 , Issue 4 (December 2004), Pages: 335 - 371, ISSN:0360-0300

[Tiwana2003] Tiwana, Amrit; Affinity to infinity in peer-to-peer knowledge platforms; Communications of the ACM; ISSN:0001-0782

[Topley2002] Topley, Kim, J2ME in a Nutshell; O'Reilly, March 2002

[White2002] White, James P.; Hemphill, David A.; Java 2 Micro Edition March 2002, Manning Publications Co.

[Wilson2002] Wilson, Brendon J.; JXTA ; Prentice Hall, 2002

[Yeager2002] Yeager, W; Williams, J; Secure Peer-to-Peer networking: the JXTA example; IT Professional, Volume 4, Issue 2, Mars/April 2002, page 53-57

9.2 Web

[3GPPsip] 3GPP requirements on SIP
http://www.ietf.org/internet-drafts/draft-garcia-sipping-3gpp-reqs-03.txt

[829-1991] 829-1991 IEEE Standard for Software Test Documentation from 1991
http://www.ustreas.gov/hrsolutions/docs/test/

[Acc008] Motorola Accompli 008 Features
http://developers.motorola.com/developers/wireless/global/uk/A008.html

[Bouncy] Bouncy Castle cryptography package
http://www.bouncycastle.org/

[CLDC1.1] SR-139 Connected Limited Device Configuratin 1.1
http://jcp.org/jsr/detail/139.jsp

[EJava] EmbeddedJava Application Environment
http://java.sun.com/products/embeddedjava/

[Forte] Forte for Java
http://wwws.sun.com/software/Developer-products/ffj/

[GNUpl] GNU public license
http://www.fsf.org/copyleft/gpl.html

[Gnutella] Gnutella
http://www.gnutella.com/

[Gong] Gong, Li; Project JXTA: A Technology Overview;
http://www.jxta.org/project/www/docs/TechOverview.pdf

[Graham2001] Graham, Ross Lee; Intelligent Information Systems LAB; Linköpings university;
http://www.ida.liu.se/conferences/p2p/p2p2001/hybrid.html

[Gupta2001] Gupta, Vipul; Bringing Big Security to Small Devices; 2001 JavaOne presentation;
http://playground.sun.com/~vgupta/KSSL/2246gupta.pdf

[IBM] IBM's WebSphere
http://www.embedded.oti.com/wdd/

[IBMrt] IBM's WebSphere, platforms supported
http://www.embedded.oti.com/compat/

[IEEE] Institute of Electrical and Electronics Engineers, Inc.
http://www.ieee.org/

[Insignia] Insignia
http://www.insignia.com

[iPaq] Compaq iPaq
http://www.compaq.com/products/handhelds/index.html

[IPInt] IP Surrogate Project
http://developer.jini.org/exchange/projects/surrogate/IP/index.html

[IPIntSpec] Jini Technology IP Interconnect Specification
http://developer.jini.org/exchange/projects/surrogate/IP/sa-ip.pdf

[J2ME_spec] Java Specifications Requests related to J2ME
http://jcp.org/jsr/tech/j2me.jsp

[J2ME] J2ME Homepage
http://java.sun.com/j2me/

[J2MEFAQ] J2ME Frequently asked questions
http://java.sun.com/j2me/faq.html

[J2METips] J2ME Tech Tips(2001)
http://developer.java.sun.com/developer/J2METechTips/2001/tt1015.html#tip2

[J2MEwtk] J2ME Wireless Toolkit
http://java.sun.com/products/j2mewtoolkit/

[JAIN] The JAIN APIs
http://java.sun.com/products/jain/

[JavaJXTA] Platform project, the Java 2 Standard Edition (J2SE) version of JXTA
http://platform.jxta.org/

[JavaSpace] JavaSpaces Service Specification
http://wwws.sun.com/software/jini/specs/jini1.1html/js-title.html

[JCard] Java Card Technology
http://java.sun.com/products/javacard/

[JCP] Java Community Process
http://jcp.org/

[Jini] Jini Network Technology - Sun's official Jini site
http://www.sun.com/jini

[JiniJO02] Jini@ JavaOne Webcast; March 25-29, 2002; San Francisco
http://www.jini.org/content/webcast/Apr2002/index.html

[JiniOrg] Jini.org - the Jini community's web site
http://jini.org

[JiniSA] Jini Surrogate Architecture Specification
http://developer.jini.org/exchange/projects/surrogate/

[JiniSAO] The Jini Technology Surrogate Architecture Overview
http://surrogate.jini.org/overview.pdf

[JiniSASpec] Jini Technology Surrogate Architecture Specification
http://www.jini.org/standards/sa.pdf

[JiniWD] Jini WirelessDevice Project
http://developer.jini.org/exchange/projects/wirelessdevice/

[JLCA] Java Language Conversion Assistant
http://msdn.microsoft.com/vstudio/downloads/jlca/default.asp

[JSR] Java Specification Requests
http://jcp.org/en/jsr/all

[JSR 179] Location API for J2ME
http://jcp.org/jsr/detail/179.jsp

[JSR-113] Java Speech API 2.0
http://jcp.org/jsr/detail/113.jsp

[JSR-120] JSR 120 Wireless Messaging API
http://jcp.org/jsr/detail/120.jsp

[JSR-129] JSR-129 Personal Basis Profile Specification
http://jcp.org/jsr/detail/129.jsp

[JSR-134] JSR-134 Java Game Profile
http://jcp.org/jsr/detail/134.jsp

[JSR-135] JSR 135 Mobile Media API
http://jcp.org/jsr/detail/135.jsp

[JSR-164] JSR 164 JAIN SIMPLE Presence
http://jcp.org/jsr/detail/164.jsp

[JSR-165] JSR 165 JAIN SIMPLE Instant Messaging
http://jcp.org/jsr/detail/165.jsp

[JSR-172] JSR 172 J2ME Web Service Specification
http://jcp.org/jsr/detail/172.jsp

[JSR-177] JSR 177 Security and Trust Services API for J2ME
http://jcp.org/jsr/detail/177.jsp

[JSR-180] JSR 180 SIP API for J2ME
http://jcp.org/jsr/detail/180.jsp

[JSR-186] JSR 186 JAIN Presence
http://jcp.org/jsr/detail/186.jsp

[JSR-187] JSR 187 JAIN Instant Messaging
http://jcp.org/jsr/detail/187.jsp

[JSR-30] JSR-30 J2ME Connected, Limited Device Configuration
http://jcp.org/jsr/detail/30.jsp

[JSR-36] JSR-36 J2ME Connected Device Configuration
http://jcp.org/jsr/detail/36.jsp

[JSR-37] JSR-37 Mobile Information Device Profile
http://jcp.org/jsr/detail/37.jsp

[JSR-46] JSR 46 Foundation Profile
http://jcp.org/jsr/detail/46.jsp

[JSR-62] JSR-62 Personal Profile Specification
http://jcp.org/jsr/detail/62.jsp

[JSR-66] JSR-66 RMI Optional Package Specification Version 1.0
http://jcp.org/jsr/detail/66.jsp

[JSR-75] JSR-75 PDA Profile for the J2ME Platform
http://jcp.org/jsr/detail/75.jsp

[JSR-82] JSR 82 Java SPIs for Bluetooth
http://jcp.org/jsr/detail/82.jsp

[JSR-118] JSR-118 Mobile Information Device Profile 2.0
http://jcp.org/jsr/detail/118.jsp

[Jump] Project JumpStart;
http://www.jumpernetworks.com/jump.html

[JumpNet] Jumper Network; owner of the Jump [Jump] and JumpStart [JumpStart] project;
http://www.jumpernetworks.com/

[JumpStart] Project Jump;
http://www.jumpernetworks.com/jumpstrt.html

[JXME] JXTA fro J2ME Project page
http://jxme.jxta.org/

[JXTAdraft] Internet-Draft at IETF. JXTA v1.0 Protocols Specification -
http://www.ietf.org/internet-drafts/draft-duigou-jxta-protocols-00.txt

[JXTAfaq] JXTA Frequently Asked Questions
http://www.jxta.org/project/www/docs/DomainFAQ.html

[JXTASec] Sun Microsystems, Inc; Security and Project JXTA
http://www.jxta.org/project/www/docs/SecurityJXTA.PDF

[JXTASpec] JXTA v1.0 Protocols Specification
http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html

[JXTAws] JXTA web site
http://jxta.org/

[KnudMem] Jonathan Knudsen, "Understanding MIDlet Memory"
http://www.javaperformancetuning.com/tips/j2me.shtml

[Krishnan2001] Krishnan, Navaneeth; The Jxta solution to P2P;
http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta-p.html

[kSOAP] The kSOAP project - SOAP for J2ME
http://www.ksoap.org/

[kXML] The kXML project
http://www.kxml.org

[Mahmoud2000] Mahmoud, Qusay; Lorain, Nicholas; Wireless Software Design Techniques
http://wireless.java.sun.com/midp/articles/uidesign/

[Mahmoud] Mahmoud, Qusay ; "Wireless Java Security";
http://wireless.java.sun.com/midp/articles/security/

[me4se] Micro Edition for Standard Edition (me4se)
http://me4se.org/

[MIDP1.0] Mobile Information Device Profile 1.0
http://java.sun.com/products/midp/

[MIDP2.0] JSR 118 Mobile Information Device Profile 2.0
http://jcp.org/jsr/detail/118.jsp

[Minar2002] Minar, Nelson; Distributed Systems Topologies: Part 2;
http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html

[Napster] Napster
http://www.napster.com/

[NET] Microsoft .NET site
http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000519

[NETcfaq] Frequently Asked Questions About the Microsoft .NET Compact Framework
http://msdn.microsoft.com/vstudio/device/compactfaq.asp

[NETcomp] The .NET Compact Framwork Overview
http://msdn.microsoft.com/vstudio/device/compactfx.asp

[NETfaq] Microsoft .NET Frequently Asked Questions (FAQ)
http://msdn.microsoft.com/library/default.asp?URL=/library/techart/faq111700.htm

[NETfto] Microsoft .NET Framework Technical Overview
http://www.gotdotnet.com/team/framework/Dot-Net%20Framework%20Technical%20Overview%20v3.doc

[NETJ#] Microsoft Visual J# .NET Beta 2
http://msdn.microsoft.com/visualj/jsharp/beta.asp

[NETJava] .NET and Java
http://www.gotdotnet.com/team/java/

[NETov] .NET Framework Product Overview
http://msdn.microsoft.com/netframework/prodinfo/overview.asp

[Pats] Program for Advanced Telecom Services (PATS)
http://www.item.ntnu.no/avantel/pats.html

[PJava] PersonalJava
http://java.sun.com/products/personaljava/

[Pnglib] Png library from sixlegs.com
http://www.sixlegs.com/software/png/

[PsiNaptic] PsiNaptic
http://www.psinaptic.com/

[PureTLS] Claymore Systems - the developer of Pure TLS
http://www.claymoresystems.com/

[RFC2396] RFC2396 Uniform Resource Indicator (URI): Generic Syntax
http://www.ietf.org/rfc/rfc2396.txt

[RFC2543] SIP: Session Initiation Protocol
http://www.ietf.org/internet-drafts/draft-ietf-sip-rfc2543bis-09.txt

[RFC2616] RFC2616 Hypertext Transfer Protocol - HTTP 1.1
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[Servlet] Tecnologia Servlet
http://java.sun.com/products/servlet/

[SETI] SETI@Home
http://setiathome.berkeley.edu/

[SIMPLE] SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE); Includes 4 internet drafts
http://www.ietf.org/ids.by.wg/simple.html

[SIP] Session Initiation Protocol (SIP); Includes 25 internet drafts
http://www.ietf.org/ids.by.wg/sip.html

[SIPPING] Session Initiation Proposal Investigation (SIPPING); Includes 12 internet drafts
http://www.ietf.org/ids.by.wg/sipping.html

[SOAP] Simple Object Access protocol (SOAP) Protocol Specification v. 1.1
http://www.w3.org/TR/SOAP

[Spotless] The Spotless System
http://research.sun.com/spotless/

[Sun] Sun Microsystems, Inc.
http://www.sun.com/

[SWaba] Superwaba
http://www.superwaba.org/

[TLS] Transport Layer Security (TLS)
http://www.ietf.org/rfc/rfc2246.txt

[TomCat] Utilização do TomCat
http://www.moreservlets.com/Using-Tomcat-4.html

[Topley] Topley, Kim; Building Wireless Web Clients: Pitfalls of MIPD HTTP, Part 1
http://www.onjava.com/pub/a/onjava/2002/04/17/j2me.html

[UDDI] Universal Description, Discovery and Integration (UDDI)
http://www.uddi.com/

[Usab] Site sobre usabilidade
http://www.usabilityfirst.com

[Vasu] Vasudevan, Venu; A Web Service Primer;
http://www.xml.com/lpt/a/2001/04/04/webservices/index.html

[W3Cws] W3C Web Services Activity web site.
http://www.w3.org/2002/ws/

[Waba] Waba
http://www.wabasoft.com

[WSDL] Web Service Definition Language (WSDL) Specification 1.0
http://www.w3.org/TR/wsdl

[WV] Wireless Village
http://www.wireless-village.org/

[XML] Extensible Markup Language (XML)
http://www.w3.org/XML/

[XP] Extreme Programming
http://www.extremeprogramming.org/

Anexo A - O Golfe

Origem

A palavra golfe provém do inglês *golf* que, por sua vez, vem do alemão *kolb*, que significa taco. Existem várias versões sobre a origem deste desporto. Uma das mais prováveis é que os escoceses o tenham criado por volta de 1.400. Já em 1457, o parlamento escocês, por ordem do rei James II, proibia a prática do golfe, por considerá-lo um divertimento que afectava os interesses do país, devido à dedicação e ao tempo que este desporto exigia.

Outras origens são conhecidas, desde o jogo romano chamado *paganica*, praticado nos séculos XVII e XVIII, em que se utilizava uma bola de couro e uma vara curva.

Há ainda os que acreditam que o golfe saiu do *jeu de mail*, antigo jogo francês que se assemelha ao golfe, mas que é praticado em espaços fechados.

As regras do golfe, tal como são conhecidas hoje, foram definidas no século XVIII, no ano de 1744, na cidade de Edimburgo, na Escócia.

Consiste em sair de um local determinado, em campo aberto, e embocar a bola no menor número de tacadas possível, em buracos estrategicamente colocados em distâncias variadas.

O jogo normalmente é disputado em percursos de 18 buracos e, numa competição, quem totalizar o menor número de tacadas no término dos 18 buracos é o vencedor.

Equipamentos

Para se jogar golfe é necessário possuir uma taqueira (saco com conjunto de 14 tacos), bolas e sapatos com sola de travas para dar firmeza no posicionamento e golpes do jogador. Opcionalmente, pode ser utilizada uma luva para evitar que o taco escorregue nas mãos.

As roupas devem ser confortáveis para propiciar liberdade de movimentos. Modernamente, os tacos são feitos de materiais leves como o carbono, mas mantâm a definição de tacos de madeira (*wood*) e tacos de ferro (*iron*), materiais que remontam à origem do desporto.

Os tacos *wood* são utilizados para tacadas de longa distância e menor precisão enquanto os tacos *iron* são apropriados para jogadas de aproximação e maior precisão.

Há ainda os tacos especiais, como o *sand*, para tirar a bola de bancos de areia; o *pitch* e o *putter*, para embocar a bola na região do *green*.

Jogo

O golfe pode ser jogado individualmente ou em grupos de dois a quatro jogadores, e tem como particularidade a ausência de um "adversário" propriamente dito; o único adversário do golfista é o próprio campo, uma vez que não há nada que ele possa fazer no sentido de dificultar o desempenho de outros jogadores.

O resultado depende do seu esforço individual e sorte, e cada golfista luta para diminuir a sua pontuação total no campo.

Em competições oficiais, é proibido ao golfista falar com outros jogadores acerca do jogo. Já em jogos entre amigos, é normal o golfista mais experiente dar "dicas" aos menos experientes.

Campo

Não há um campo de golfe igual a outro. Por isso, cada campo é um novo desafio.

Há campos no meio de desertos; em regiões montanhosas; em planícies; em regiões costeiras, etc.

Os campos normalmente são formados por conjuntos de 9 ou 18 buracos. Um campo de golfe oficial ocupa cerca de 1 milhão de metros quadrados.

O percurso total de 18 buracos, geralmente, tem cerca de seis quilómetros de extensão em linha recta e demora perto de quatro horas e meia para ser concluído. Por essa razão, o golfe é um excelente desporto para promover a socialização e fazer amigos.

Não é raro multinacionais incluírem nas suas exigências profissionais questões referentes à prática de golfe por parte do candidato a emprego, uma vez que grandes negócios são fechados em campos de golfe.

Projecto

Os principais campos de golfe do mundo são desenhados por grandes projectistas, como os estilistas de moda, contribuindo para a fama do campo.

Cada buraco é planeado para testar a habilidade dos jogadores - há buracos considerados fáceis e outros difíceis, conforme a complexidade.

O buraco pode estar atrás de uma curva, em cima de um monte, no meio de um lago, ou simplesmente, num campo aberto de fácil acesso.

Normalmente no *tee* (local de saída) há um mapa para que o jogador prepare a melhor estratégia para chegar ao buraco.

Natureza

Vale a pena ressaltar que o golfe é um desporto ecológico pois coloca o homem em contacto com a natureza e estimula a preservação de árvores e animais, favorece a socialização e não tem idade para ser praticado, podendo ser jogado por jovens, adultos e crianças.

Handicap

O golfe também possibilita que jogadores mais e menos experientes possam disputar uma partida entre si, através do sistema *handicap*, que são tacadas de bonificação dadas ao jogador menos experiente para serem descontadas no término do jogo.

Quanto menor o *handicap* melhor o jogador. Um profissional tem *handicap* zero.

O *handicap* varia de 0 a 24 para homens e de 0 a 36 para mulheres e, conforme o jogador progride, vai "baixando" o *handicap*, até chegar a zero, prosseguindo como amador ou profissional.

Percurso

O percurso de um buraco visto por cima apresenta os seguintes elementos: *tee*, local de saída; *fairway*, região de relva baixa onde é fácil para o jogador dar a próxima tacada; *rough*, região de relva alta, onde é difícil bater a bola; e o *green*, local de relva rasteira e muito aparada, com altura média de 2 mm, onde fica o buraco.

O *fairway* pode ser entrecortado por rios, lagos, bancas de areia (*bunkers*) e outros obstáculos (*hazards*) para dificultar o progresso do jogador.

Par do buraco

Desde o local de saída (*tee*) até ao buraco, o número médio de tacadas necessárias para embocar a bola é um índice, chamado par do buraco, que ajuda a medir o desempenho do jogador. Conforme a distância, há buracos de par três (até 228 m), par quatro (até 430 m) ou par cinco (mais de 430 m).

Para as mulheres as distâncias são um pouco menores. É nos buracos de par três que o golfista tenta a famosa jogada *hole-in-one*, que consiste em embocar a bola em apenas uma jogada.

Par de campo

A somatória total do par de todos os buracos dá origem a outro valor de referência, o par de campo.

Se um campo tem par 71, quer dizer que um jogador regular deve, ao fim dos 18 buracos, totalizar o mais próximo possível de 71 tacadas.

Quando o par é 71 e o jogador termina os 18 buracos em 70 tacadas, é comum dizer que fez "um abaixo do par"; se terminou com 69, "dois abaixo do par", e assim por diante. No mesmo para 71, se marcar 72, diz-se "um acima do par"; 73, "dois acima do par", e assim sucessivamente.

Modalidade

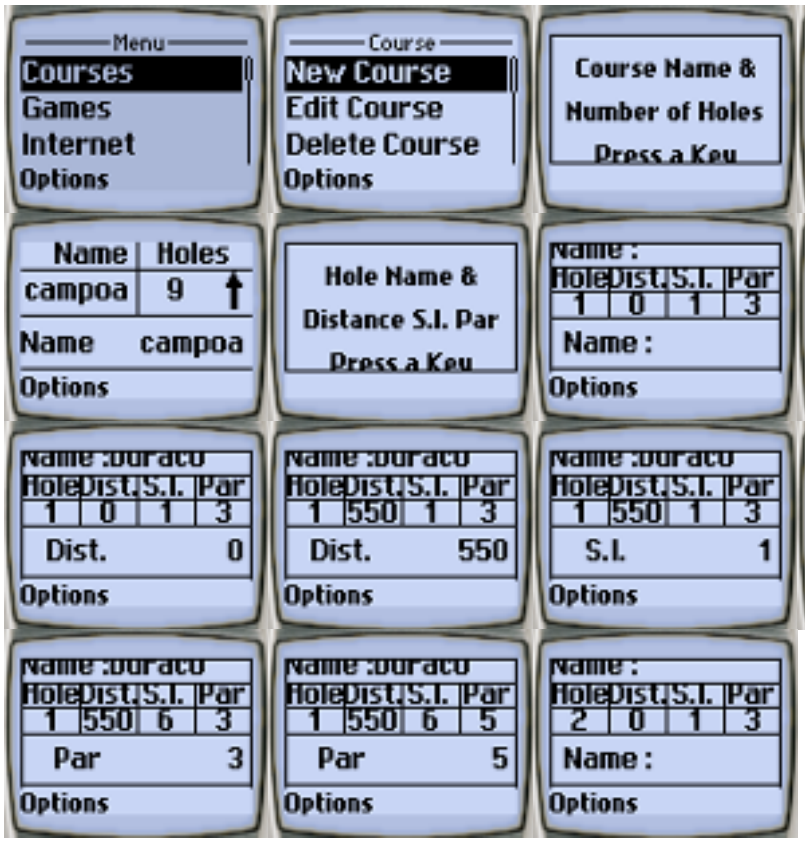
Há muitas modalidades de jogo de golfe.

As mais utilizadas são *stroke play*, onde o vencedor é definido pelo menor número na somatória das tacadas; e *match play*, onde são conferidos pontos a cada buraco - por exemplo, no primeiro buraco o jogador A emboca a bola em menos tacadas que o jogador B, recebendo um ponto; e assim por diante. Quem somar o maior número de pontos é o campeão.

O golfe também pode ser jogado em duplas ou trios, com a somatória do resultado de cada jogador da equipa; duplas mistas; e uma infinidade de variações.

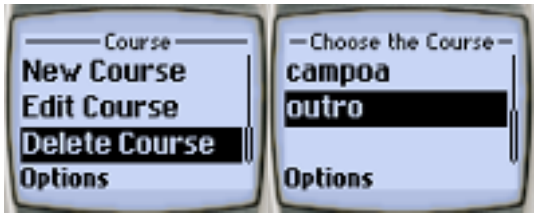
Anexo B – *Layout* dos ecrãs do mGolfScoreCard com sequência de navegação

Inserir (ou editar) um novo campo de golfe

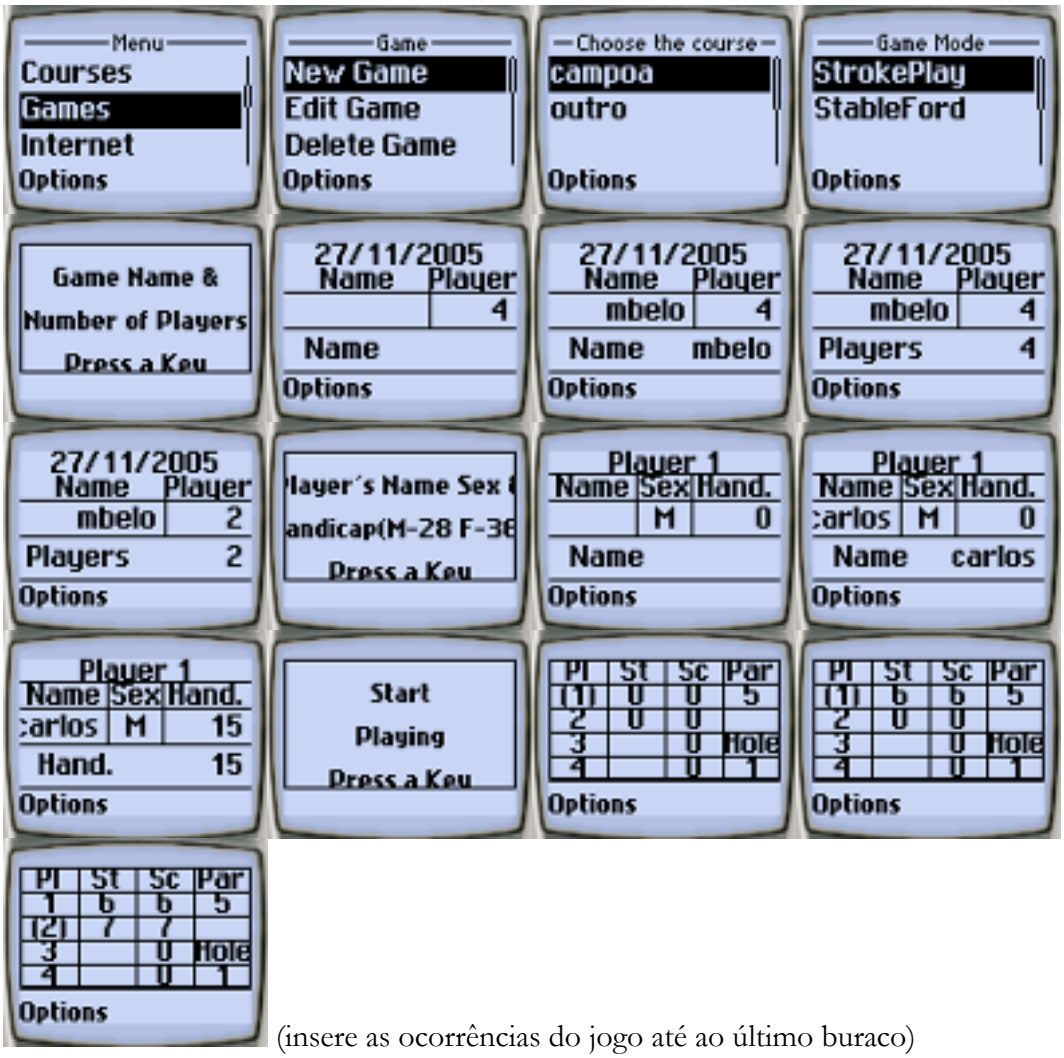


(...até ao último buraco)

Remover um campo de golfe

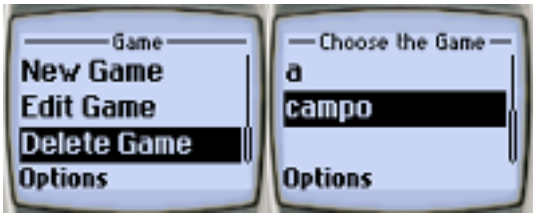


Realizar um novo jogo de golfe



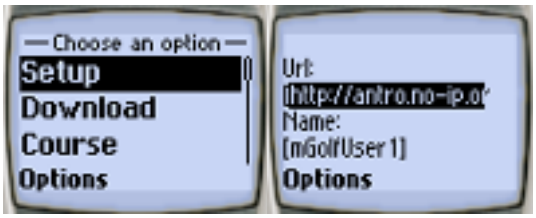
(insere as ocorrências do jogo até ao último buraco)

Remover um jogo de golfe



Aceder à zona de partilha P2P

Configurar o endereço do servidor *relay* e nome do utilizador



Aceder aos campos partilhados

Partilhar campos

Ajuda

Anexo C - Teste de Usabilidade

Nome: _____

Idade: _____

Sexo: ❑ masculino

❑ feminino

Área Profissional: _____

Há quanto tempo joga golfe? _____

Dispositivo móvel usado: _____ Tem: ❑ teclado físico ❑ touch screen

Tarefas:

1 – Veja qual é o par do buraco 7 do campo "mbelo".

_____

2 – Qual foi o resultado final do jogo "verao".

_____

3 – Altere o nome do jogador 3 do jogo "verão" de "joaquim" para "manuel".

_____

4 – Insira um campo de golfe chamado "leiria" com valores arbitrários.

_____

5 – Simule um jogo entre a "maria" e a "ana" no campo "leiria".

_____

6 – Partilhe o campo "leiria".

_____

7 – Veja quais os campos actualmente partilhados por outros jogadores.

_____

Observações:

_____

_____

_____

Aspecto Gráfico/Visual:	Mau	Fraco	Suficiente	Bom	Muito Bom
Menus					
Cores					
Qualidade gráfica					
Legibilidade					

Funcionalidade:	Mau	Fraco	Suficiente	Bom	Muito Bom
Inserção de campos					
Edição de campos					
Remoção de campos					
Criação de um jogo					
Edição de um jogo					
Remoção de um jogo					
Partilha de campos					

	Mau	Fraco	Suficiente	Bom	Muito Bom
Facilidade geral de utilização					

Utilidade para o jogo de golfe	Mau	Fraco	Suficiente	Bom	Muito Bom
Controlo da classificação do jogo					
Comunicação entre jogadores					
Acompanhamento *off-online* do jogo					
Utilidade geral para o jogo de golfe					

	Mau	Fraco	Suficiente	Bom	Muito Bom
Apreciação global					

Perguntas:

O que achou da introdução de dados na aplicação através de *wizards*?

_____

Considera possível o utilizador "perder-se" na aplicação?

_____

Sentiu a necessidade de consultar o manual do utilizador? Se sim, achou que o seu conteúdo tem alguma utilidade e está bem explícito ou, pelo contrário, necessitou de ajuda complementar?

_____

O que gostou mais nesta aplicação?

_____

Gostaria de ver alguma coisa alterada ou melhorada na aplicação?

_____

Anexo D – Manual do Utilizador

mGolfScoreCard
v1.0

António Carlos Alves Urbano
© 2005

Introdução

Obrigado por aquirir o mGolfScoreCard. Esperamos que ao utilizar o mGolfScoreCard, disfrute de muitas horas de prazer e entretenimento.

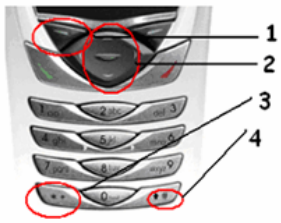
O mGolfScoreCard é a ferramenta ideal para qualquer jogador num campo de golfe. Substituindo o tradicional cartão de papel, permitindo jogar nos mais famosos modos de golfe e calcular a pontuação final. Esperemos, por isso, que seja do seu agrado.

2

Tabela de Conteúdos

3

Funcionalidades do Teclado



1-Soft buttons: servem para navegar através dos menus
2-Botões direccionais: permitem navegar e mudar valores nalguns menus
3-Tecla *: serve para apagar todos os caracteres inseridos
4-Tecla #: serve para apagar carácter a carácter

4

Modos de Jogo e Significado das Siglas usadas

Nesta aplicação pode-se jogar nos modos de jogo mais populares entre os jogadores de golfe, o StableFord**(1)** e o StrokePlay**(2)**.

1-No modo de jogo StableFord existe uma tabela de pontuação pelo número de tacadas dadas por buraco comparativamente ao respectivo par(*), sendo descontado o *handicap*(**) através do Stroke Index(***).

2-No modo de jogo StrokePlay, cada tacada um ponto. No final retira-se o *handicap*.

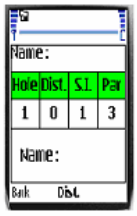
(*)**Par** – Valor médio de tacadas a atingir em cada buraco e que serve para atribuir uma pontuação.
(**)Handicap**(Hand.)** – Valor que cada jogador tem, quanto mais baixo melhor jogador.
(***)Stroke Index**(SI)** - Indice atribuído a cada buraco que vai de 1 até ao número de buracos

5

Inserir um Campo

Nome e Nº Buracos: aqui pode inserir o nome e nº buracos: sem mudar de menu pressione as teclas **Cima** e **Baixo** para mudar o nº de buracos(9/18)

Neste menu pode navegar horizontalmente usando as teclas ◄— e —►.



S.I. : Para introduzir este valor deve utilizar as teclas **Cima** e **Baixo**

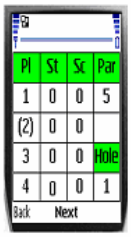
Par : Na introdução do Par use apenas as teclas com os valores permitidos para o Par (3,4,5).

6

Inserir um Jogo

Nome: Depois de inserir o nome, pressione as teclas **Cima** e **Baixo** até o valor para Sexo ser o pretendido

Handicap: O Handicap para homens varia entre 0 e 28 e para mulheres entre de 0 a 36.



Strokes(Nº Tacadas): neste menu insere o nº de tacadas, valor que é inserido para o jogador seleccionado (o que está entre parêntesis)

Pontuação(Score): está ao lado nº de tacadas

Nº Buraco(Hole) : no canto inferior direito

Par: Situa-se no canto superior direito.

A navegação faz-se com as teclas **Cima** e **Baixo**

7

Configuração da Internet e descarregar um campo

Menu Internet/Setup

Url : Basta inserir um URL com o endereço de um servidor válido

Name : Nome com que se gostaria de identificar para uma partilha ou descarga de um campo.

Menu Internet/Download

Neste menu terá acesso à lista de campos disponíveis para descarga. Seleccionando o campo e aguardando que a barra de progresso chegue ao fim, o processo estará finalizado.

8

Partilhar um Campo

Basta seleccionar o campo desejado e continuar.

Durante a partilha, poderá visualizar a partilha do campo pela barra de progresso, para sair basta pressionar o botão correspondente.

9