limite texto

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Davide Ricardo Baptista Pereira

UbiShare - Partilha Interativa de Conteúdos em Ambientes Ubíquos

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor José Francisco Monteiro Morgado

Outubro de 2012

Aos meus pais.

Agradecimentos

Desde já, quero agradecer ao Dr. Francisco Morgado pelo facto de ter aceite ser o Orientador deste projeto e por me dar indicações precisas e de valor para o desenvolvimento da tese.

Ao João Abrantes, que com os seus conhecimentos/ideias técnicas e não só, teve um papel bastante importante, imprescindível e fulcral desde o início do projeto.

A toda a equipa da Load-Interactive, pois estiveram sempre disponíveis para contribuírem com ideias e sugestões durante este processo.

Um especial agradecimento à minha família, pois fizeram, sempre, uma "especial força" para que eu chegasse a esta fase do meu percurso académico. A sua persistência/ajuda foi, também, fulcral nesta caminhada.

Finalmente um muito obrigado a todos os que de alguma maneira contribuíram direta ou indiretamente neste projeto tornando-o possível.

Resumo

Hoje em dia o uso exponencial de dispositivos móveis e de rede sem fios, fazem com que a comunicação ubíqua seja uma área emergente de pesquisa. Esta pode representar uma evolução significativa na área da tecnologia da informação, apoderando-se, assim, do quotidiano das pessoas.

Apesar dos diferentes domínios da computação ubíqua, destacamos a adaptação de conteúdos/aplicações entre dispositivos, como sendo o nosso principal foco de pesquisa.

Neste trabalho, pretende-se criar uma linguagem única que faça a integração de dispositivos móveis, utilizando mecanismos que permitem a partilha de documentos através de interfaces alternativas. Cria-se assim, uma nova introdução ao paradigma de interação entre utilizadores, capaz de possibilitar que o dispositivo pessoal do utilizador, se envolva numa rede de dispositivos heterogéneos, partilhando vários formatos de documentos.

Para que esta integração seja possível, o dispositivo do utilizador deve detetar os vários dispositivos disponíveis através de sistemas de localização, podendo o utilizador, intuitivamente e através de um movimento direcional (por exemplo: *finger swipe*) executar/enviar uma tarefa para o dispositivo pretendido. Na *Cloud* ficam armazenados os documentos referentes a cada utilizador, de modo a que este possa partilhá-los posteriormente, com outros utilizadores que utilizem o mesmo sistema.

Palavras-chave: realidade aumentada, computação ubíqua, dispositivos móveis, interação, cloud, partilha, webservices, XML, iOS

Abstract

Nowadays, the exponential use of mobile devices in wireless network environments makes ubiquitous communication an emerging field of research. This may represent a significant evolution in the information technologies area, seizing, thus, people's every day.

Despite the different fields of ubiquitous computing, we highlight the adaptation of content / applications between devices, as our main research focus.

With our work, we aim at creating a unique language that makes the integration of mobile devices, using mechanisms that allow content sharing via alternative interfaces. This creates a new introduction to the paradigm of interaction between users, enabling device to be involved in a network of heterogeneous devices, sharing content with various document formats seamlessly.

For this integration to become possible, the device should detect the user's various devices available through tracking systems, and the user can intuitively, through directional movement (eg finger swipe) share content with another device. In the Cloud, content is stored for each user so that one can subsequently share the same content again with other users.

Keywords: augmented reality, ubiquitous computing, mobile devices, interaction, cloud, share, web services, XML, iOS

Índice

Índice de tabelas

Índice de figuras

Acrónimos

AoA	Angle of Arrival
AP	Access Point (Ponto de Acesso)
API	Application Programming Interface (Interface de Programação de Aplicações)
AR	Augmented Reality (Realidade Aumentada)
BS	Base Station (Estação Base)
Cloud Computing	Nuvem de Computação
GMD	German National Research Center for Information Technology
GPS	Global Positioning System (Sistema de Posicionamento Global)
HCI	Human-computer interaction (Interação Humano-Computador)
HTML	HyperText Markup Language (Linguagem de Marcação de Hipertexto)
HTTP	HyperText Transfer Protocol (Protocolo de Transferência de Hipertexto)
IaaS	Infrastructure-as-a-Service (Infraestrutura como um Serviço)
IDE	Integrated Development Enviroment (Ambiente Integrado de Desenvolvimento)
IEEE	Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Eletricistas e Eletrónicos)
IPSI	Integrated Publication and Information Systems Institute
iRoom	Interactive Room
iROS	Interactive Room Operating System
LOS	Line of Sight
MVC	Model-View-Controller
NLOS	Non Line of Sight
OS	Operating System (Sistema Operativo)
PaaS	Plataform-as-a-Service (Plataforma como um Serviço)
PHP	Personal Home Page
RF	Rádio Frequência

RSSI	Received Signal Strength Indicator (Indicador de Força de Sinal recebido)
RSS	Received Signal Strength (Força de Sinal Recebido)
SaaS	Software-as-a-Service (Software como um Serviço)
SDK	Software Development Kit
SGBD	Sistema de Gestão de Base de Dados
SS	Signal Strength
SOAP	Simple Object Access Protocol (Protocolo Simples de Acesso a Objetos)
TDoA	Time Difference of Arrival
ToA	Time of Arrival
Xerox PARC	Centro de Pesquisa de Tecnologia da Xerox Corp. em Palo Alto-CA-EUA
XML	Extensible Markup Language
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Network

1. Introdução

Este capítulo visa introduzir e contextualizar a interação entre dispositivos através do sistema abordado, assim como dar a conhecer os objetivos do trabalho desenvolvido. Será também discutido neste capítulo, os motivos que levaram à escolha deste tópico.

A primeira parte do capítulo incidirá sobre o contexto do projeto, mostrando como o projeto poderá ser útil no quotidiano das pessoas.

A seção do propósito indica-nos qual é o objetivo que se pretende alcançar com a realização deste estudo, traçando também, as principais metas deste.

A seção de motivação visa explicar o porquê da escolha desta temática e da relevância da mesma no panorama atual.

A seção de estrutura do documento dar-nos-á uma visão global do documento, bem como uma fácil compreensão da organização do mesmo.

1.1. Contexto

O título tem como objetivo dar a entender de uma forma sucinta do que esta dissertação se trata.

Cada vez mais a utilização de dispositivos móveis é uma alternativa ao uso dos computadores de secretária que eram, quase, a única solução disponível para ter acesso à informação.

Uma série de projetos têm investigado o uso de dispositivos móveis para interagir com o mundo real [6,7,10]. Esses projetos utilizam diversas técnicas de interação tais como tocar ou apontar, e também métodos de comunicação diferentes (infravermelhos,

Bluetooth, Wi-Fi, etc.). Entretanto, os mais recentes dispositivos móveis integram novas tecnologias (tais como acelerómetros, bússolas digitais e ecrãs sensíveis ao toque) que potenciam possibilidades de interação adicionais.

As redes sem fios são vistas como um grande potencial de comunicação tendo mesmo vindo a ser desenvolvidas um vasto número de aplicações, sendo a disponibilização de conteúdos baseados na localização de utilizadores uma delas [24] [25], como é exemplo o museu Pergamon de Berlim, onde são disponibilizados aos utilizadores PDAs que lhes mostram unicamente a informação referente à obra que se encontra na sua proximidade.

Aproveitando as potencialidades dos dispositivos móveis e das redes sem fios, pretendemos alimentar o paradigma de interação do utilizador com ambiente, fazendo assim, que seja possível a troca de informação/conteúdos entre dispositivos, que se encontram na mesma rede e que, utilizem o mesmo sistema para o efeito. Esta interação será feita através de um método de reconhecimento do gesto direcional de arrastamento (*finger swipe)*. E apesar de não ser o nosso foco, o sistema de localização de dispositivos móveis na rede é fulcral nesta caminhada.

1.2. Propósito

O objetivo principal deste projeto é proporcionar ao utilizador um recurso simples, mas útil, que possa ser utilizado em qualquer situação do dia-a-dia, tanto em ambientes profissionais, bem como pessoais.

Após feita uma revisão do estado da arte, podemos dizer que a abordagem elaborada nesta dissertação, torna-se em algo diferente, com características também diferentes das existentes, pois face ao estudo e uso de vários recursos, pretende criar um novo conceito de interação com o utilizador.

Numa forma introdutória podemos dizer que o nosso estudo pretende proporcionar ao utilizador funcionalidades desde a localização de dispositivos até à partilha de conteúdos (imagens, ficheiros pdf, vídeos, músicas, links, …) através, mas não só, de movimentos direcionais em tempo real.

As principais metas são:

- Fornecer uma prova de conceito estável e confiável, capaz de realizar as funcionalidades chave.
- Acesso do Utilizador ao sistema mediante certas normas de segurança;
- Localização de dispositivos móveis
- Upload de ficheiros para a *cloud*
- Partilha de conteúdos com outros utilizadores
- Apresentar o conteúdo partilhado, através de *plugins* incorporados (*pdf reader, vídeo player, web browser, music player, image player*)

1.3. Motivação

Os sistemas móveis e ubíquos constituem hoje uma área de investigação bastante abrangente e cuja importância tem vindo a crescer rapidamente e exemplo disso é o crescimento acentuado do uso dos dispositivos móveis, pois estes a cada dia que passa tornam-se mais poderosos, mais tecnológicos, possibilitando o desenvolvimento de novas aplicações, de novos serviços e formas de interação.

Contudo, a computação ubíqua e o desenvolvimento de aplicações para dispositivos móveis ainda não encontraram o seu estado maduro, devido à limitação de soluções abertas e genéricas.

Pretende-se, com este trabalho, que haja uma conceção, implementação e teste de uma solução para dispositivos *iOS*, que utiliza algoritmos de localização de dispositivos móveis e também a informação proveniente de sensores que estes dispositivos

integram (acelerómetros, bússola digital e ecrãs sensíveis ao toque) para interagirem entre si, partilhando informação/conteúdos.

Esta dissertação teve igualmente o objetivo de criar *know-how* na área de desenvolvimento para dispositivos móveis, em sistemas de localização e também sobre a ubiquidade.

Assim sendo, podemos dizer que a motivação para esta tese é contribuir para que o desenvolvimento para dispositivos móveis dê mais um passo rumo à maturidade.

1.4. Estrutura da dissertação

O presente documento encontra-se estruturado em 5 capítulos.

O primeiro (e atual) é o capítulo introdutório, onde se enquadra o tema e se apresenta o contexto onde o trabalho realizado se insere. São apresentadas, posteriormente, as motivações a que levaram e impulsionaram o desenvolvimento desta dissertação, bem como a organização do mesmo.

O segundo capítulo – "Estado da Arte" – tem como objetivo fornecer uma análise ao contexto atual, que assenta em 3 partes: análise sobre computação móvel e ubíqua; localização de dispositivos em redes sem fios; armazenamento e troca de informação. E também, uma perceção de onde o projeto pode ser efetivamente uma mais-valia.

O terceiro capítulo – "Desenvolvimento" – trata-se do desenvolvimento propriamente dito do projeto, apresentado os resultados obtidos, bem como as ferramentas utilizadas para o efeito.

O quarto capítulo – "Conclusão" – irá analisar o trabalho desenvolvido, sobre os efeitos da atual dissertação, qual o trabalho futuro possível para a continuação da mesma.

O quinto capítulo – "Referências Bibliográficas" – contém todas as referências bibliográficas que serviram de apoio ao desenvolvimento desta dissertação.

O sexto capítulo (e último) – "Anexo" – contém blocos de código que descrevem métodos aplicados na elaboração do protótipo.

2. Estado da Arte

Este capítulo fornece uma compreensão mais detalhada do que já existe apontando diretrizes para o futuro.

2.1. Computação Móvel e Ubíqua

A evidente evolução das tecnologias, das redes sem fio e de dispositivos móveis, levou à Computação Móvel e Ubíqua.

A computação móvel faz-nos referência a aspetos de mobilidade, desde mobilidade do utilizador, com a possibilidade deste comunicar a qualquer hora, em qualquer lado, com quem quiser (ex: ler/escrever um email através de um aplicativo ou de um web browser), até à portabilidade do dispositivo, uma vez que os dispositivos podem ser ligados a qualquer hora, a qualquer altura a uma determinada rede. Porém existe uma importante limitação quando há uma alteração da posição física, isto é, quando o utilizador se move para um outro contexto, que é a inalterabilidade do modelo computacional. Como exemplo, podemos referenciar a situação em que o utilizador está numa sala de espetáculos, onde o seu dispositivo se encontra, à partida, em modo silencioso, e quando este se dirige para fora dela, precisa novamente de ajustar o seu dispositivo. Esta necessidade do utilizador alterar/reconfigurar o estado não é aceitável nesta área de estudo.

A computação móvel tem como requisitos, sistemas de *middleware*, dispositivos móveis e redes sem fios (ex: *Infravermelhos*, *Bluetooth*, *Wi-Fi,* etc) - área cuja evolução tem sido bastante importante no suporte à mobilidade. Mas também de dispositivos computacionais distribuídos no ambiente que sejam percetíveis ou não, tais como sensores, reconhecimento de voz, etc., denominando-se esta como computação *"pervasiva"*.

Esta ideia, de Computação Ubíqua (também denominada de Ubicomp), surge através do cientista Mark Weiser e da sua equipa Xerox PARC na década de 90 (abordaremos alguns protótipos desenvolvidos), devido à necessidade de integração da computação com o mundo/espaço físico (ex: paredes, móveis, carros, xícaras de café, canetas etc). Este refere-se ao termo Computação Ubíqua, como um contexto onde a presença computacional num objeto é totalmente transparente para quem o usa e em alguns dos cenários totalmente invisível, dando a escrita como sendo a primeira tecnologia que se tornou ubíqua em países industrializados, dizendo que as tecnologias que são mais profundas são aquelas que "desaparecem" [1].

No início da relação entre computadores e humanos, muitas pessoas partilhavam um único computador para fazer uso dos seus serviços, então a ideia de que cada utilizador pode ter o seu próprio computador alterou significativamente a forma como as pessoas usavam os sistemas computacionais. Na última década esta ideia foi-se transformando para a relação de muitos-para-um, onde cada utilizador tem muitos computadores, ou pelo menos, dispositivos com capacidade de processamento. Porém, a Computação Ubíqua remete-nos para alguns problemas, para além da relação qualitativa entre computador e utilizador, sendo a privacidade e a segurança exemplo disso [1].

A Computação Ubíqua afeta o nosso dia-a-dia, apesar de muitos não se darem conta disso. Hoje temos, Hotspots WiFi em lugares como cafés, bibliotecas, restaurantes, centros comerciais, etc. Telemóveis com internet 3G, serviços de voz sobre IP, tendo talvez o Skype como um dos softwares mais usados para o efeito. Mesmo os automóveis já disponibilizam ao condutor sistemas de alta tecnologia, como são o caso dos computadores de bordo.

E as casas inteligentes, conceito que é hoje muito falado, onde toda a casa está interligada, permitindo coisas fantásticas, desde o acender de luzes por uma instrução vocal ou, imagem só, pelos frigoríficos que estão encarregues de enviar informação/pedidos de compra ao supermercado, quando detetam que se está a esvaziar ou, a nossa torradeira que sabe que estamos na hora do pequeno-almoço e

nos prepara umas torradas, onde a cafeteira faz também parte do processo sem nós precisarmos de lhes tocar sequer nem dizer nada. A isto chamamos de Computação Pervasiva que é uma espécie de subárea da Computação Ubíqua.

2.1.1. Projetos da UbiComp da Xerox

A partir dos anos 90 foram desenvolvidos alguns protótipos de UbiComp, por parte do Xerox PARC, conforme podemos verificar na Figura 1.

Figura 1 - Protótipos desenvolvidos pelo Xerox PARC [4]

O *PARCtab* é um dispositivo portátil sensível ao toque, gravando a informação selecionada, onde este comunicava com outros dispositivos via infra-vermelhos.
O *PARCpad* era um notebook fixo e sem mobilidade com caneta e microfone, com comunicação por rádio.
O *Liveboard* é um conceito utilizado hoje em dia nas escolas, pois é um dispositivo com um ecrã sensível ao toque que grava as informações à medida que interagimos com o mesmo através de uma caneta.

Através de um vídeo do youtube [26], podemos ver uma pequena demonstração destes protótipos.

2.1.2. Projeto Microsoft Easyliving

Para além das casas inteligentes, como mencionamos anteriormente, podemos também falar dos prédios inteligentes.
A Microsoft foi quem desenvolveu os primeiros esforços para conceber um prédio inteligente, com o projeto chamado *Microsoft Easyliving* através de um grupo de pesquisa sobre Computação Ubíqua, inserida na empresa.

O objetivo passava por desenvolver uma arquitetura e tecnologias em ambientes inteligentes que facilitavam a interação desimpedida de pessoas com outras pessoas, com computadores e com dispositivos. A tecnologia pretendia permitir que a pessoa pudesse efetuar chamadas a partir de qualquer ponto da casa, simplesmente com o uso ao reconhecimento de voz. Pretendia também controlar as crianças e animais domésticos automaticamente e que quando a pessoa passasse pelas várias divisórias da mesma (dados obtidos de sensores), era mantida uma sessão interativa com o computador [3].

O *Easyliving* tinha também utilidade em laboratórios, pois, quando uma pessoa entrava nele recebia uma identificação provisória, podendo aceder a um computador ou dispositivo com uma senha ou scanner biométrico. Se eventualmente, a pessoa mudasse de sala e acedesse a outro dispositivo, automaticamente era transferida a informação do antigo display para o novo do computador em uso [4].

Na Figura 2 temos uma ilustração presente no artigo [4] onde existem, num escritório, 3 hotspots. A interface do utilizador migra entre os 3 monitores, pois este movimenta-se de hotspot em hotspot [4].

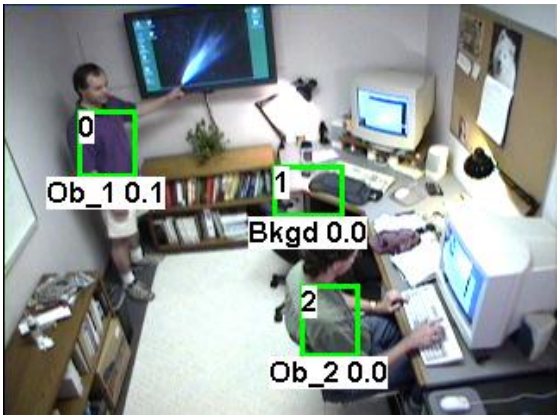

Figura 2 - Espaço de testes do Easyliving [3]

2.1.3. Projeto AURA

No projeto *Aura*, foram desenvolvidos arquiteturas de sistema, algoritmos, interfaces e técnicas de avaliação que visam minimizar distrações do utilizador, criando um ambiente que se adapta ao contexto e às suas necessidades. O Aura assenta em componentes/sistemas já existentes, como o *Coda* e o *Odyssey*, que foram sendo adaptados/alterados de maneira a atender às necessidades da computação pervasiva.

O *Odyssey* suporta monitorização de recursos e adaptação de aplicações, enquanto o *Coda* fornece suporte para o acesso a ficheiros com adaptação de rede. Depois, temos o *Spectra* que é um mecanismo de execução remota que usa o contexto para decidir a melhor forma de executar a chamada remota. Por fim, temos o *Prism* que é responsável pela captura e gestão das intenções do utilizador [7]. Na Figura 3 podemos ver o esquema da arquitetura do Aura.

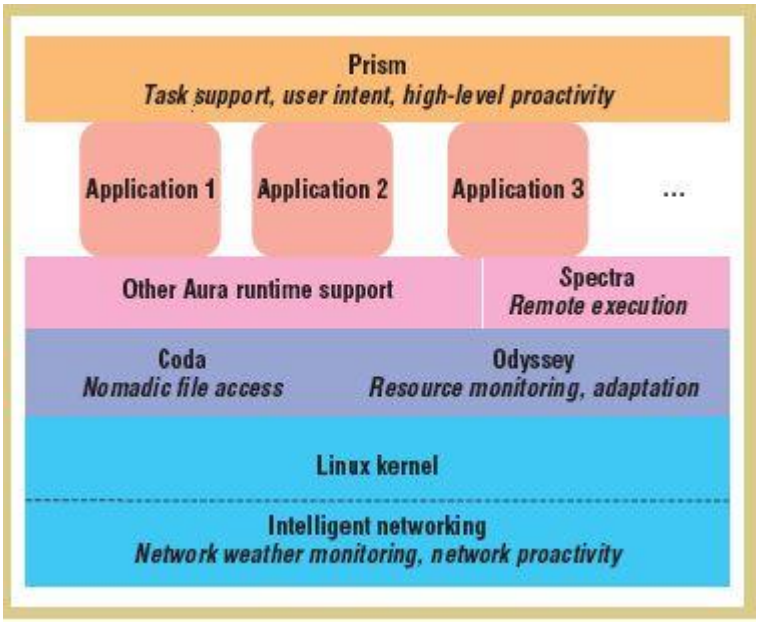

Figura 3 - Arquitetura do Aura [44]

2.1.4. Projeto Gaia

O projeto *Gaia*, desenvolvido na Universidade de Illinois, pretende ser um sistema operativo, que visa apoiar o desenvolvimento e a execução de aplicativos portáteis para espaços ativos. Gaia é uma infraestrutura *middleware* distribuída que coordena entidades de *software* e redes heterógenas contidas num espaço físico, exportando serviços para consultar e utilizar os recursos existentes, de modo a permitir o acesso e o uso do contexto atual, oferecendo uma *framework* para desenvolvimento centrado-no-utilizador, recurso-consciente, multi-dispositivo, sensível ao contexto e a aplicações móveis. [5-6].

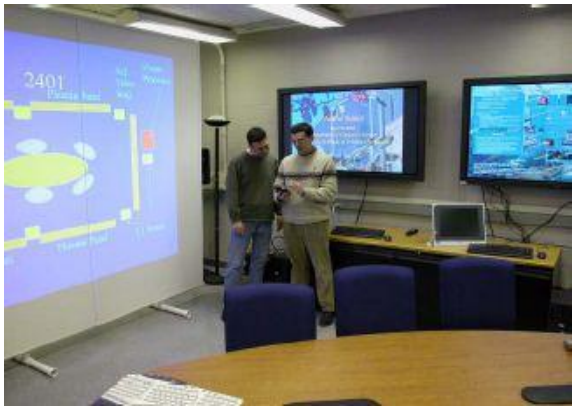

Figura 4 - Espaço ativo de testes do Gaia [6]

2.1.5. Projeto Interactive Workspaces

O projeto *Interactive Workspaces*, iniciado na Universidade de Stanford, é um projeto que investiga a conceção e utilização de saldas contendo vários ecrãs de alta resolução, com capacidade para integrar dispositivos portáteis e de criar aplicações que integram a utilização de vários dispositivos no espaço interativo [7].

Figura 5 - Vista da Sala Interativa (iRoom) [7]

Na Figura 5, podemos observar o espaço interativo utilizado, chamado *iRoom*, uma infraestrutura de *software* para esse ambiente, chamado *iROS*, e experiências de interação humano-computador (*HCI*) no espaço.

A *iROS* possui 3 subsistemas: o *Event Heap*, que é o mecanismo que coordena as interações das aplicações em execução e os demais subsistemas do iROS, armazenando e enviando mensagens, conhecidas como “eventos”, onde cada uma é uma coleção de campos nome-tipo-valor; o *Data Heap* que facilita a movimentação dos dados no espaço interativo; o *iCrafter*, que fornece um sistema para “anúncios/publicidade” e invocação de serviços [7].

2.1.6. Projeto Roomware

O projeto *Roomware*, proveniente do *GMD-IPSI*, consiste na integração de elementos aumentados (ex: paredes, portas, móveis com mesas, cadeiras, etc) com dispositivos computacionais de informação (cooperative buildings) tornando-os interativos, fornecendo suporte para a criação, edição e apresentação de informações. Utilizando uma infraestrutura de rede onde os componentes estão ligados na mesma, permite acesso em toda a parte. As cadeiras e a mesa são móveis devido às redes sem fios independentemente dos fornecimento da energia. Utiliza uma infraestrutura de software que permite que as pessoas comuniquem e partilhem de informação, onde mediante uma infraestrutura de localização e sensorial, estas podem fazê-lo em qualquer parte, permitindo a mobilidade dentro do ambiente. Assim sendo o objetivo principal do *Roomware* é permitir o avanço para a integração dos espaços arquitetónicos e espaços de informação. Introduzindo novas formas de interação humano-computador para multiutilizadores, multi-monitores e ambientes. [8-9]

2.1.7. Aplicação Doodle Space

O *Doodle Space* é uma aplicação interativa desenvolvida para ecrãs de grandes dimensões, que permite que vários participantes de forma colaborativa (em conjunto), pintem curvas *3D* numa parede projetada, usando dispositivos com câmaras. É usado um algoritmo baseado em sequência de imagens para estimar os parâmetros do movimento das câmaras. Permite também, que o utilizador controle o pincel, com os movimentos da câmara, rodando, arrastando e aumentando o tamanho das curvas. É utilizado um mecanismo de transmissão sem fio *Bluetooth* [10]. Nas seguintes figuras podemos ver a arquitetura de comunicação do *Doodle Space* e também a interação de múltiplos utilizadores com a aplicação.

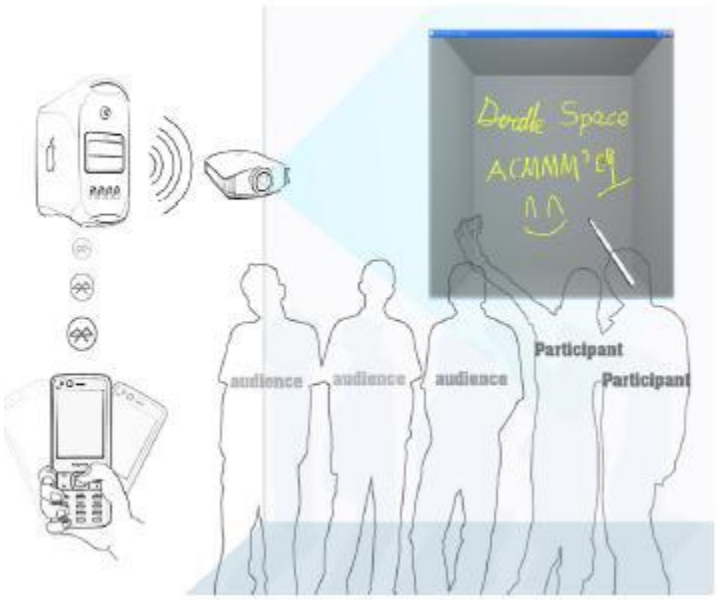

Figura 6 - Arquitetura de comunicação do Doodle Space [10]

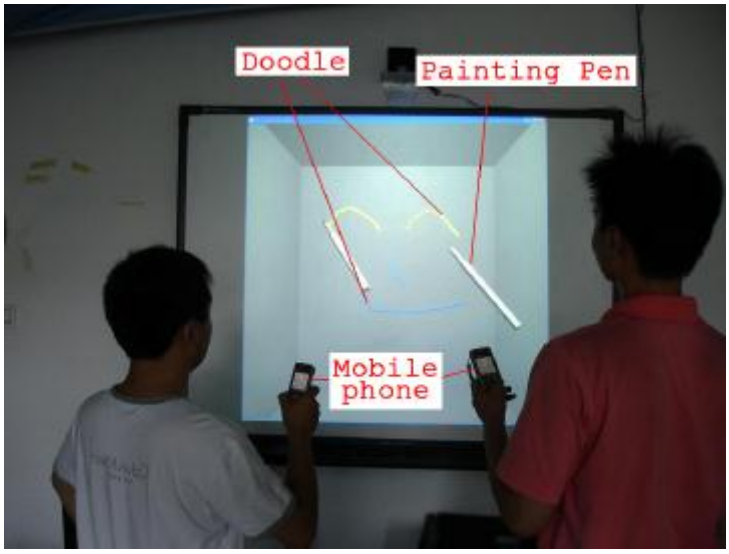

Figura 7 - Múltiplos participantes desenhando através do Doodle Space [10]

2.1.8. Conclusão

Apesar de ainda não termos chegado ao ponto que Weiser menciona, caminhamos para esse "novo mundo"/futuro que este aponta, dizendo que não é necessária nenhuma revolução na inteligência artificial, mas apenas incorporar a computação no quotidiano [1].

2.2. Sistemas de localização em redes sem fios

Com os consequentes avanços e desafios da Computação Ubíqua, o cálculo automático da localização de um determinado dispositivo móvel numa rede sem fios, torna-se cada vez mais necessário. A localização de um dispositivo móvel é um parâmetro crítico nas aplicações baseadas no contexto, devido à necessidade de precisão na estimativa de localização. A exploração dos sinais em redes Wi-Fi tem sido a alternativa mais utilizada para resolver problemas de localização em ambientes fechados, como prédios, escritórios, centros comerciais, casas, etc. Estes ambientes têm como característica o fato de serem menos fiáveis do que as redes de cabos, originando assim, taxas de erro superiores. Este tipo de redes normalmente são construídas para cobrir apenas uma zona em particular, como por exemplo, um escritório ou uma escola, por isso é normal haver algumas quebras quando o utilizador se move para uma determinada zona onde há pouca cobertura de sinal. Por isso, as aplicações móveis têm que estar preparadas para acompanhar a disponibilidade dos serviços de rede e de reagir de forma adequada [11].

Neste subcapítulo apenas iremos abordar alguns sistemas de localização em redes Wi-Fi.

2.2.1. RADAR

O primeiro passo nos sistemas de localização, foi dado pelo projeto *RADAR*, desenvolvido pela Microsoft Research, que é um sistema baseado em radiofrequência (RF) para localização e monitorização de utilizadores dentro de edifícios, usando o padrão 802.11 para medir a potência do sinal em várias estações base devidamente posicionadas, fornecendo cobertura numa determinada área. A intensidade do sinal RF é usada como medida de distância entre o emissor RF e o recetor, sendo que esta informação é usada para localizar o utilizador recorrendo a técnicas de triangulação [12].

Para determinar a informação da posição, o *RADAR* usava duas fases:

- A fase offline: o utilizador indica a sua localização atual, clicando num mapa, onde as coordenadas (x,y), orientação (d) e o timestamp (t) são registadas, sendo que cada combinação de localização e orientação eram guardados num tuplo com o formato (t, x, y, d). O tuplo poderia ser interpretado como uma previsão da potência do sinal recebido por cada estação fixa [12].

- A fase online: Aqui, existem duas abordagens para determinar a localização em tempo real (online). O método empírico que permite comparar o SS analisado com o gravado na base de dados e escolhe a melhor combinação para estimativa de localização. E o de propagação RF, onde o número de paredes entre o recetor e emissor são fatores levados em conta para calcular a distância dos sinais.

Utilizando a última abordagem (a online), o sistema preserva a privacidade do utilizador. Mais propriamente, através do método empírico foram obtidos os melhores resultados, com um erro de 1.92, 2.94, 4.69 metros em 25, 50 e 75% das localizações [15].

Dadas as experiências, os investigadores verificam que numa dada posição, a potência do sinal varia. A orientação do utilizador numa determinada posição influencia, também, a potência do sinal Wi-Fi recebido [14].

Na Figura 8, podemos ver uma relação entre distância e potência de sinais Wi-Fi recebidos pelos AP (3).

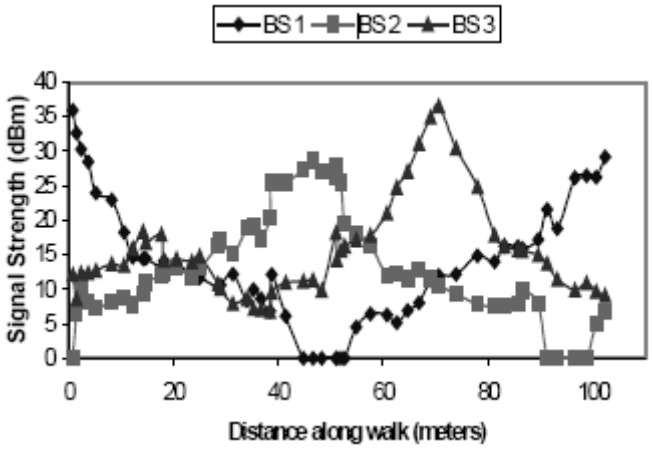

Figura 8 - A força do sinal em três estações [12]

2.2.2. Ekahau

O *Ekahau RTLS*, desenvolvido pela empresa Finlandesa *Ekahau*, também se baseia em padrões de redes sem fios 802.11 para fornecer um sistema de localização em tempo real (RTLS) para localizar dispositivos e pessoas, também em tempo real.

Com o *Ekahau* não há interrupção das atividades ou comunicações da rede. As posições dos dispositivos e pessoas são atualizados automaticamente e podem ser entregues a outros sistemas que necessitem dessa informação [16].

É um sistema de localização com boa precisão, pois a localização dos dispositivos móveis em Wi-Fi estão ao alcance de menos de 1m, sendo este o erro médio. Como ponto a favor, tem o fato de não ser preciso um grande poder computacional para localizar os dispositivos, pois a localização é processada através de um dispositivo central, aumentando assim a fiabilidade da mesma.

Na Figura 9, podemos ver a arquitetura do *Ekahau*.

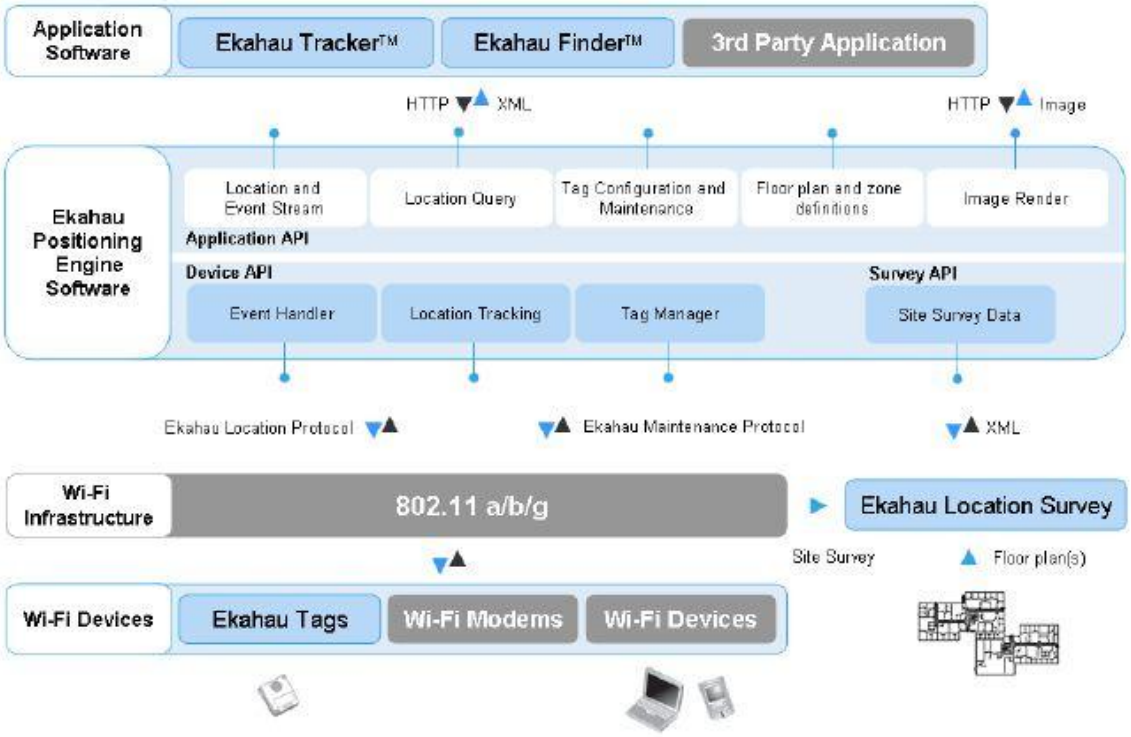

Figura 9 - Arquitetura do Ekahau [16]

Ekahau RTLS é uma solução de *tracking* inteiramente baseado nas movimentas de pessoas. Este consiste na integração de vários componentes, como podemos ver na Figura 9 [16].

O *Ekahau Positioning Engine*, atualmente na versão 4.4, é o cérebro do *Ekahau RTLS*. Este é um serviço web que corre em cima de um *Windows Server* dedicado [16]. Fornece uma interface web de fácil manuseamento para definição como podemos verificar na Figura 10.

Figura 10 - Interface Web do Ekahau Positioning Engine [16]

O *Ekahau* é dos poucos a ser atualmente comercializado, tendo alcançado bons resultados em ambientes hospitalares, na localização de *stocks* e na indústria automóvel [14].

2.2.3. Horus

O *Horus*, desenvolvido na Universidade de Maryland em 2004, visa satisfazer dois objetivos, o da alta precisão e o de baixos requisitos computacionais. É implementada no contexto de redes sem fio 802.11, onde permite a localização de um grande número de dispositivos móveis [17]. A alta precisão a que consegue chegar deve-se em grande parte à identificação que o *Horus* faz às diferentes causas da variação dos sinais Wi-Fi e como ele as consegue resolver [14].

O sistema utiliza a intensidade do sinal observado pelas *frames* transmitidas através de pontos de acesso (*AP*) permitindo a localização do utilizador.
Como é um sistema instalado no dispositivo do cliente e não no servidor, abrange uma maior número de clientes, apesar de se tornar menos cómodo para o utilizador.

A partir do *Horus*, podem ser implementadas algumas aplicações, tais como, entrega de conteúdo na localização, procura de direções, controlo de ativos e notificações.

Este assenta em duas fases, uma fase offline de testes e calibrações, e uma fase online de determinação da localização. Durante a fase offline, o sistema guarda a intensidade do sinal recebido a partir dos *AP's* localizados em áreas de interesse, de modo a possibilitar a construção de um mapa rádio. Durante a fase de determinação da localização, o sistema usa as amostras da potência de sinal recebido a partir dos AP, para "procurar" o mapa radio e estimar a localização do utilizador.

O mapa radio é baseado nas técnicas deterministas e probabilísticas. As primeiras, representam a intensidade do sinal de um *AP* num determinado local, apresentando um valor médio. Por outro lado, as técnicas probabilísticas armazenam a informação sobre a distribuição dos *AP* no mapa rádio, usando as técnicas para estimar o local do utilizador.

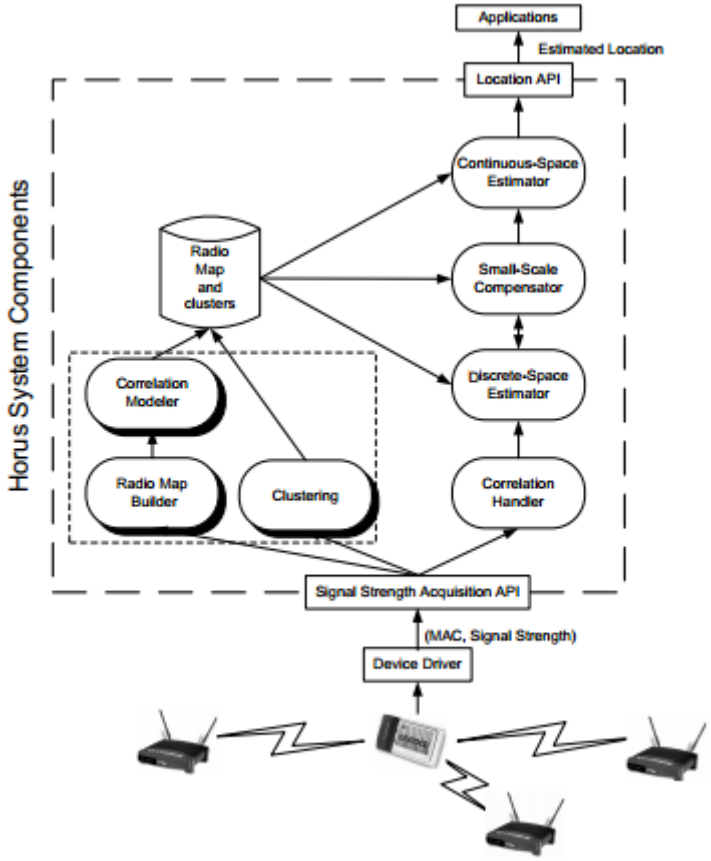

Figura 11- Componentes do Sistema Horus [17]

Na Figura 11, podemos ver os componentes do sistema *Horus*, onde as setas representam o fluxo da informação no sistema. Os blocos sombreados representam os módulos utilizados durante a fase offline [17].

Assim, o *Horus*, reduz os requisitos computacionais do algoritmo (*lightweight*) de determinação da localização, através da aplicação das técnicas *location-clustering.*

2.2.4. WBLS [18]

O *WBLS*, desenvolvido na Universidade de São Paulo, é uma variação do sistema básico de localização, onde a novidade reside na forma de cálculo das probabilidades de observação. A sua intenção é a de reduzir, através da eliminação de uma certa quantidade de informação, uma fonte de ruído nessa mesma informação que é fornecida ao sistema de localização, com o objetivo de se obter estimativas de localização de melhor qualidade.

O sistema procura lidar com o fato de que a informação obtida nas medições realizadas, para a construção do mapa de RSSI sobre as frequências de presença dos sinais, pode não ser confiável.

O *WBLS* é, considerado pelo autor, como simples, pois no cálculo das probabilidades da observação para cada um dos estados, em vez de se realizar um produto das probabilidades relativas a cada um dos *AP* do ambiente, passam a ser consideradas apenas as probabilidades relativas aos sinais presentes, fazendo com que, para cada *AP* seja calculada uma probabilidade tendo a presença do sinal como condicionante, o que implica uma alteração na forma do cálculo.

Este sistema tem também capacidade de lidar com o problema de emissão de sensores, pois este somente considera os sinais presentes nas observações, fazendo assim, que não sofra muitas penalizações quando um dos emissores é desligado.

2.2.5. Comparação entre sistemas de localização em redes sem fios

Na Tabela 1 apresentamos sistemas de localização indoor para redes Wi-Fi, de modo a ser possível uma visão geral dos mesmos.

Sistema	Tecnologia	Algoritmo	Exatidão	Precisão
RADAR	WLAN (RSSI)	kNN	3 a 5m	50% a 2.94m 75% a 4.69m
Ekahau	WLAN (RSSI)	Método probabilístico	<1m	50% em 2m
Horus	WLAN (RSSI)	Método probabilístico	1.4m	90% em 2.1m
Univ. Ray Juan Carlos				-
WBLS	WLAN (RSSI)	Método probabilístico	1.5m	-

Tabela 1 - Comparação entre sistemas de localização em redes Wi-Fi

Estes sistemas, baseados na tecnologia *Wi-Fi*, têm a vantagem de estarem bastante inseridos nas residências, provendo assim a infraestrutura para sistemas de localização. Tanto esta, como a tecnologia *Bluetooth* (não abordada), não foram especialmente desenvolvidas para baixo consumo energético, ao contrário da tecnologia *ZigBee IEEE 802.15.4* (não abordada), pois não necessita de elevada taxa de transmissão, sendo ideal para redes de sensores, tornando-se, talvez, a tecnologia apropriada para sistemas de localização de baixo custo, com elevada durabilidade mas que não necessitem de uma exatidão elevada [15].

2.3. Técnicas de localização

Os sistemas de localização podem ser categorizados segundo diferentes técnicas, as quais são usadas para estimar a localização dos dispositivos móveis [15], sendo que cada uma delas tem a sua vantagem e desvantagem, adequando-se a um ou a outro contexto. Sendo assim, iremos aborda-las para termos uma melhor noção de qual técnica se adapta melhor ao contexto de localização Wi-Fi.

A localização é possível determinar utilizando uma infraestrutura para o posicionamento, como é o caso do GPS, ou modificando de alguma forma o sistema de comunicação utilizado [19].
O *GPS* é uma técnica que, apesar de ser a mais popular em ambientes externos, se torna ineficiente em várias aplicações, onde o desempenho se concentra em ambientes indoor (fechados). Para além de que, o *GPS* necessita de *hardware* específico, o que o inviabiliza em diversas aplicações devido ao custo.

Por isso, deve-se utilizar técnicas que já tenham implementado um sistema de comunicação sem fios em particular aquelas que seguem o padrão 802.11, para localização de dispositivos dentro deste mesmo padrão.

Muitas das técnicas utilizadas para a localização com sinais de radiofrequência consistem na realização de alguma forma da triangulação [18]. Esta técnica baseia-se nas propriedades geométricas dos triângulos e pode ser baseada numa técnica baseada na distância (*lateration*) ou na diferença angular (*angulation*) entre dispositivos [15].

2.3.1. Cell-ID

É uma técnica simples para determinar a posição de uma determinada estação. O *AP* com maior sinal assume-se como sendo a posição da estação.

Esta técnica não faz qualquer tentativa explícita para resolver a posição de um dispositivo móvel para além da indicação da célula com a qual o dispositivo móvel é registado [22].

Em geral a precisão desta técnica depende de quão elevada a densidade da célula é. Em áreas rurais, um dispositivo móvel, normalmente cobre uma área de até 30km, enquanto em cidades este valor reduz-se para 10metros apenas.

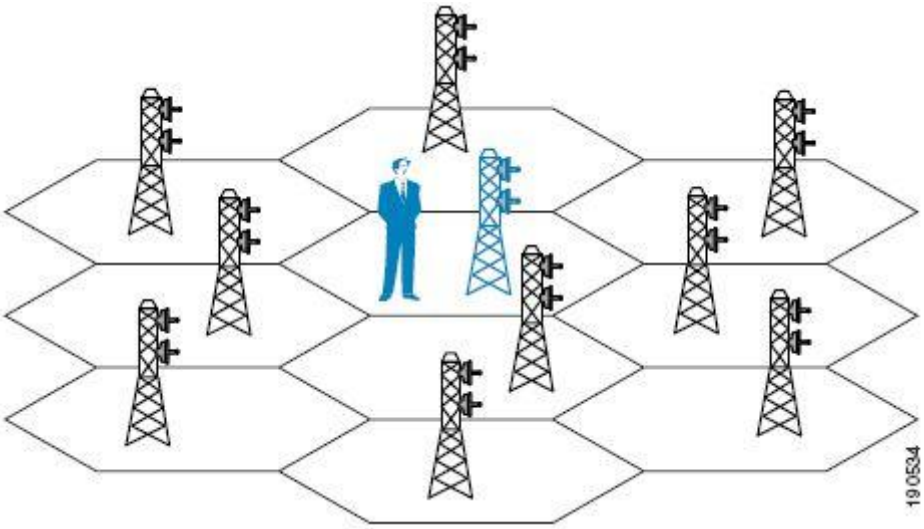

Figura 12 - Localização baseada em Cell-ID [22]

2.3.2. AoA

AoA (*Angle of Arrival*) é uma técnica que utiliza o ângulo de chegada do sinal *RF* e que para estimar a posição, utiliza de pelo menos 2 pontos de referência no raio de acesso ao dispositivo, calculando o angulo e o tamanho das arestas do triângulo formado. Pode-se aumentar a precisão com o acréscimo de bases recetoras. Basicamente, as antenas direcionais são ligadas ao recetor e a área à volta do recetor é testada para medir e marcar a partir de qual direção o sinal recebido é mais forte.

O desempenho dos métodos de localização baseados nos ângulos de chegada, são limitados pela precisão das medidas, sendo esta uma característica do *hardware* e do algoritmo de estimação [20].

Figura 13 - Localização baseada em AoA

Na Figura 13, a antena do dispositivo móvel recebe sinal e determina o ângulo de onde o sinal chega. A posição pode ser calculada através da combinação dos ângulos dos sinais das duas bases.

Sistemas que utilizem o *AoA* não servem para ambientes *indoor*, devido aos múltiplos caminhos de propagação *NLOS* que um sinal pode percorrer até ser captado pela antena do aparelho recetor. Além disso, a implementação de um sistema que utiliza *AoA* necessita de antenas direcionais nos recetores, o que não é viável para localização *Wi-Fi* [18].

2.3.3. ToA

ToA (*Time of Arrival*) é uma técnica que mede o tempo de chegada de um determinado sinal entre o emissor e o recetor. Com base na cronometragem do tempo de percurso entre o emissor do sinal e a sua receção no dispositivo recetor é possível calcular a distância entre os dois dispositivos [15].

Ao contrário da *AoA*, as técnicas baseados no tempo não se degradam com o aumento da distância entre o dispositivo e as estações base (*BS*), sendo mais fácil obter melhor precisão utilizando estas medidas em vez das de ângulo, apesar de ser necessária uma linha direta visível entre a fonte e as *BSs*, para que isso seja possível [20].

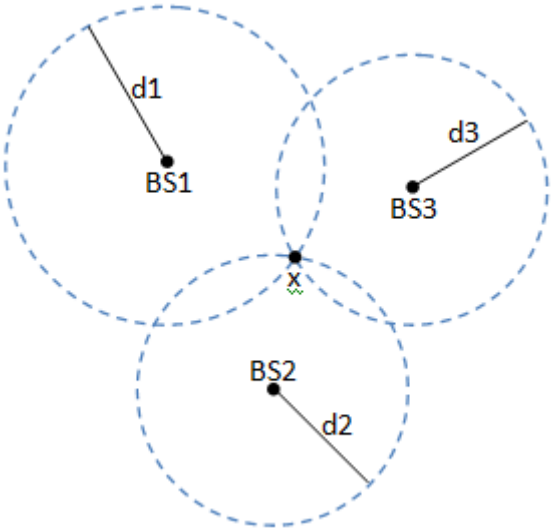

Figura 14 - Localização baseada em ToA

Na Figura 14, a interseção entre as 3 circunferências (x), é a localização do dispositivo. Ou seja, este método implica que existam pelo menos 3 estações base (*BS*) para se poder saber a posição exata do dispositivo que se quer localizar.

Conforme foi mencionado em cima, esta técnica precisa de saber exatamente o tempo que o sinal radio é emitido e recebido, sendo este o principal problema desta, pois qualquer que seja o desvio no tempo medido, irá afetar a distância calculada, aumentado assim o erro da localização.

Sistemas que utilizam esta técnica são, por exemplo, o *Active Bat* que é uma continuação do *Active Badge* [21] e o *GPS*.

2.3.4. TDoA

TDoA (*Time Difference of Arrival*) é uma técnica onde o tempo de propagação do sinal de RF é medido e este valor é utilizado para estimar a posição/distância do emissor [19]. Esta técnica é uma ligeira adaptação da *ToA*, que mede a diferença de tempo de um sinal emitido a partir da *BS* do cliente (dispositivo) para vários. Esta difere da *ToA*,

pois usa valores de tempos relativos do tempo de chegada, enquanto a *ToA* usa valores de tempo absolutos, tonando-se assim mais fiável.

Esta técnica requer recetores especiais nas *BSs* e relógios precisos e sincronizados para a monitorização dos tempos de chegada. Quando é emitido um sinal através do dispositivo móvel, para as *BSs*, os tempos de chegada são medidos em cada célula correspondente à BS.

Um sinal é emitido pelo cliente (dispositivo móvel) e sendo recebido em 3 ou mais *BSs* em tempos ligeiramente diferentes. Estes desvios de tempo acontecem devido às diferentes distâncias que separam o recetor a partir das várias BSs. Usando uma das diferenças de tempo pode-se calcular as várias localizações possíveis para o cliente. Este grupo de localizações possíveis, tem a forma de hipérbole.

Na Figura 15 demostra-se o uso da mesma, onde a interseção de 3 hiperboles é usada para determinadar a posição do dispositivo movel (x).

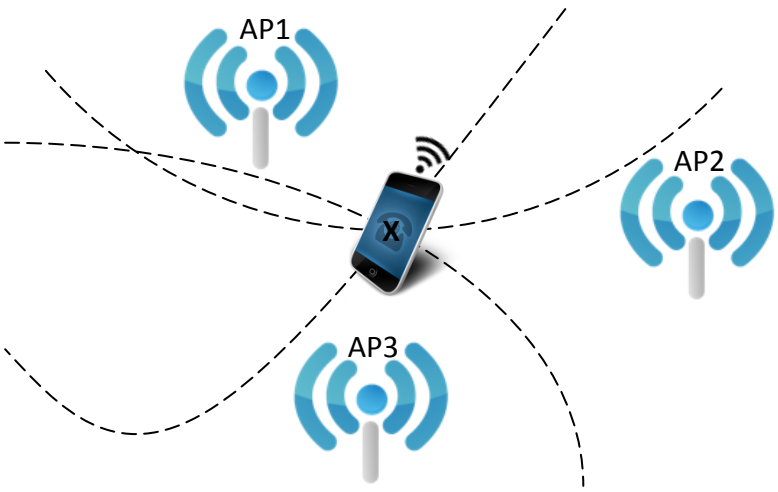

Figura 15 – Localização baseada em TDoA

De referir que esta técnica teve, como primeira aplicação, um trabalho desenvolvido em 2005 sobre localização [23] e foi testada numa rede Wi-Fi IEEE 802.11b, sendo que os autores referem que o bom posicionamento geográfico dos *AP* são muito importantes para melhorar os resultados desta técnica.

2.3.5. RF fingerprinting

A localização *Radiofrequência fingerprinting* não faz uso das medidas de tempo, ao contrário da *ToA* ou *TDoA*, embora a fusão destas duas técnicas de localização seja possível. *RF fingerprinting* utiliza o *RSSI* para calcular a localização do dispositivo em vez do tempo, sendo que com nesta abordagem o *RSSI* é medido por qualquer dispositivo móvel ou recetor de sensores.

As implementações utilizando *RSSI* têm desfrutado de uma vantagem de custo uma vez que os dispositivos não necessitam de *hardware* específico. Logo, sistemas de localização baseados em *RF fingerprinting* são normalmente fáceis de implementar e rentáveis.

Numa fase inicial há um mapeamento do ambiente, para posteriormente comparar, em tempo real, com uma base de dados pré-armazenada. Esta fase é denominada de calibragem.
Devido a efeitos de propagação, como perda, várias medições são efetuadas para minimizar este efeito. Depois de haver um conhecimento de toda a área, o sistema de localização está pronto para responder a pedidos de localização. Quando um pedido é solicitado, são efetuadas novas medições e são comparadas com os resultados obtidos na base de calibragem/calibração. A essência desta técnica prende-se como os dados são comparados e a localização é calculada.

Num local onde existem vários pisos, cada um destes é encarado como sendo distinto em termos de assinaturas RF, tornando-os únicos para que o sistema de localização possa dar resultados precisos.

Esta é uma técnica que não necessita de *hardware* adicional, como o *TDoA* ou *ToA*, sendo baseada em soluções de software quando na maioria das vezes as medidas de RSSI se tornam suficientes para fornecer resultados precisos.

2.3.6. Comparação entre as técnicas

Na tabela 2 apresentamos técnicas de localização para redes Wi-Fi, de modo a ser possível uma visão geral dos mesmos.

Técnica	**Perda da Confiabilidade**		**Complexidade do Access Point**
	Multi-percurso	Distância (AP-STA)	
ToA (TDoA)	Pouca	Nenhuma	Média
AoA	Pouca	Pouca	Alta
RSSI	Severa	Pouca	Baixa
Cell-ID	Nenhuma	Severa	Baixa

Tabela 2 - Comparação entre técnicas de localização [22]

Ainda sobre estas técnicas podemos dizer que o *ToA/TDoA* têm como vantagem a escalabilidade, elevada exatidão para condições *LOS* e que não necessita de um treino exaustivo. Mas como desvantagem apresentam elevados requisitos de sincronização, com um *hardware* complexo (logo preço mais elevado), necessitando de maiores requisitos de largura de banda, tornando-se muito sensível a condições *NLOS*.

Quanto à técnica *AoA*, esta somente necessita de 2 medidas para localização a *2D* e 3 para *3D*, não necessitando de elevados requisitos de sincronização e de treino exaustivo. Porém esta técnica apresenta como desvantagem o fato de também necessitar de *hardware* complexo e de ter elevados problemas em condições *NLOS*.

Quanto à técnica *RSSI* tem como vantagem o fato de usar *hardware* simples, sem custos adicionais de sincronismo, com maior resistência para condições *NLOS* e uma menor sensibilidade à largura de banda. Porém apresenta como desvantagens a menor exatidão comparativamente com as técnicas anteriores, isto em condições *LOS*. Também necessitam de elevado tempo de treino e de algoritmos de elevada complexidade, não apresentado assim, elevada escalabilidade [15].

Quanto à técnica *Cell-ID*, a principal vantagem é a facilidade de aplicação, pois não necessita de algoritmos complexos e assim o desempenho do posicionamento é bastante rápido. Porém, a principal desvantagem centra-se na *granularidade*, pois por vezes os dispositivos podem ser associados a células que não estão na proximidade física, apesar das outras células vizinhas poderem ser melhores. Com isto, a tentativa de resolução da localização de um dispositivo móvel numa estrutura que tenha vários andares, pode ser bastante frustrante, pois há uma considerável sobreposição do piso até ao piso onde se encontra o dispositivo [22].

Atualmente, quando se fala em sistemas de localização de dispositivos sem fio em *WLANs*, o que se deseja são mecanismos capazes de estimar a posição do utilizador num ambiente onde se utiliza equipamentos e *hardware* (ex. pontos de acesso e placas sem fio) usualmente encontrados no mercado, sem a utilização de hardware adicional, com o menor custo possível e com o mínimo esforço de calibragem.

2.4. Algoritmos de localização

Existem vários algoritmos de localização baseados em *fingerprinting*, entre eles: algoritmos probabilísticos, algoritmos de deteção de vizinhança mais próxima (*Nearest neighbor* [12]), redes neurais, máquina de suporte vetorial (*SVM*); e menor polígono *M-vertex (SMP)* [15].

Estes algoritmos estimam a localização baseando-se na calibração da fase offline.
Nesta fase são registadas assinaturas RF associadas a posições específicas. Esta calibração permite posteriormente relacionar os valores de RSSI detetados na fase de deteção (*online*) com as assinaturas estimando assim um valor para a posição do dispositivo localizável [15].

Ainda temos os métodos deterministas que são aqueles cujos algoritmos trabalham com variáveis deterministas em toda a sua cadeia (diferente dos métodos

probabilísticos, que trabalham com variáveis aleatórias e distribuições de probabilidade) [18].

Estes tentam encontrar a distância mínima de sinal, entre o vetor de atributos observado e cada um dos vetores das diferentes posições que constituem o mapa RSSI. Isto pode, ou não, ser igual à distância mínima entre a localização gravada na fase de calibração [22].

São métodos que primam pela simplicidade de compreensão e de implementação, mas não levam em consideração informações sobre as estatísticas das variações de RSSI, não incorporando, também, a relação de dependência entre estimativas consecutivas de localização [18]. O primeiro método determinista apresentado era simples [13].

Não faz parte do âmbito desta dissertação o estudo aprofundado sobre algoritmos de localização, deste modo foram somente mencionados alguns deles.

2.5. Armazenamento e troca de informação (Cloud Computing)

Abordamos este tópico, pois esta dissertação concentra-se, no armazenamento e na troca de informação entre dispositivos.

Neste nosso projeto para haver troca de informação, para além da localização dos dispositivos, por exemplo, necessitamos de armazenar informação acerca dos utilizadores, desde dados relativos ao acesso, bem como os seus documentos que posteriormente serão passíveis de partilha.

Pelas necessidades supracitadas, e uma vez que a computação ubíqua quer-se presente neste trabalho, vimo-nos “obrigados” a falar sobre *Cloud Computing* (Nuvem de Computação).

De uma maneira simplista podemos a abordar o tema *Cloud Computing* como sendo algo onde podemos aceder a ficheiros, dados, programas e serviços desde que haja uma ligação à internet, onde estes são alocados num determinado fornecedor, pagando apenas pelos recursos e serviços utilizados.

O "mundo da internet" já se encontra bastante enraizado no nosso dia, e o *Cloud Computing* representa um novo paradigma de mudança no que à implementação de sistemas distribuídos diz respeito, que se concentra em fornecer serviços escaláveis fazendo uso de tecnologias existentes, tais como a virtualização. Estes serviços estão tipicamente divididos em três categorias: Infraestrutura-como-um-Serviço (IaaS), Plataforma-como-um-Serviço (Paas) e Software-como-um-Serviço (SaaS).

Os tipos de tecnologia de *Cloud Computing* podem ser vistos em duas perspetivas: capacidade e acesso. Na Figura 16 podemos ver três tipos baseados na capacidade e dois baseados no acesso a recursos [27].

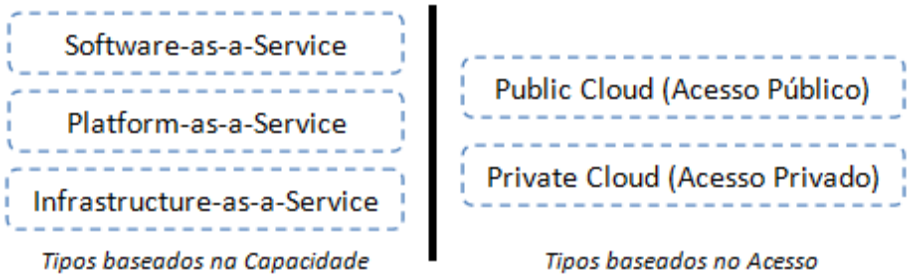

Figura 16 - *Cloud Computing* - os vários tipos baseados na capacidade e no acesso

2.5.1. Software-as-a-Service

SaaS é uma metodologia baseada em um-para-muitos, onde uma aplicação é partilhada entre vários clientes através da internet, como de um serviço se tratasse. Em vez de ser instalado e feita a compra e manutenção ao *software*, basta acede-lo através de um *browser*, não sendo necessário o fornecimento de hardware ou outro tipo de software mais complexo. Estas aplicações são executadas em servidores de um determinado fornecedor. O fornecedor disponibiliza o acesso à aplicação, incluindo

segurança, disponibilidade e também desempenho. Assim, os clientes não precisam de investir em licenças de *software* ou em servidores.

2.5.2. Platform-as-a-Service

PaaS basicamente são serviços orientados para o desenvolvimento, testes, implementação, alojamento que suporta o desenvolvimento de aplicações aos *developers*. Para estes, o *Paas* reduz os problemas e despesas adicionais com configuração e implementação das aplicações, facilmente tornando-as escaláveis.
Permite também a integração com base dados e com *web services*.

2.5.3. Infrastructure-as-a-Service

IaaS tem como objetivo o fornecimento de uma infraestrutura computacional, geralmente em ambientes virtualizados, como um serviço. Em vez do cliente comprar servidores para uma determinada aplicação, somente precisa de solicitar um *datacenter* de acordo com os requisitos da infraestrutura, tendo assim, acesso à plataforma e ao *software*. O valor deste serviço varia mediante a necessidade do cliente e conforme a sua utilização.

2.5.4. Public Cloud

Nos serviços públicos (*Public Cloud*) qualquer pessoa, com ligação à internet, pode usufruir destes. Aqui os serviços têm pouco ou nenhum controlo sobre a infraestrutura da tecnologia subjacente. No entanto, é atraente pois reduz a complexidade e os longos prazos em testes e implementação de novos produtos. Normalmente o custo é menor.

2.5.5. Private Cloud

No *Private Cloud* são oferecidos praticamente os mesmos serviços do *Public Cloud*, sendo que a diferença está nos equipamentos e sistemas utilizados para construir essa mesma nuvem, que se encontram dentro da infraestrutura da própria empresa. Ou seja, a empresa tem uma nuvem só para si, construída e mantida dentro dos seus domínios.

A necessidade de segurança e privacidade é um dos fortes motivos para que as empresas adotem este tipo de nuvem, pois estas por vezes têm dados críticos e demasiado importantes para permitirem que outra empresa responda pela sua proteção.

A vantagem é que a nuvem pode ser moldada com maior precisão às necessidades da empresa.

A desvantagem é que existe um custo elevado, pelo menos no início do processo, mas por outro lado existem benefícios que serão obtidos a medio e longo prazo que compensarão os gastos, como por exemplo, a segurança ou a agilidade dos processos.

2.5.6. Providers (Fornecedores)

O *Google Apps*, é um pacote de serviços que a *Google* oferece que conta com aplicações desde editor de texto, folha de cálculo e apresentações (*Google Docs*), ferramenta de agenda (*Google Calendar*), ferramenta de comunicação instantânea (*Google Talk*), email com o próprio domínio da empresa, armazenamento de ficheiros (*Google Drive*), entre outros.

A *Amazon* é um dos maiores serviços de correio eletrónico, onde a empresa montou uma megaestrutura de processamento e armazenamento de dados. A partir daqui, começaram a alugar este tipo de serviços/recursos, de onde resultou o serviço *Simple Storage Solution (S3)*, para armazenamento de dados e o *Elastic Compute Cloud (EC2)*, para uso de máquinas virtuais.

O *iCloud*, é um serviço da *Apple* que armazena documentos, músicas, vídeos, fotos, que têm como objetivo fazer com que as pessoas usem a nuvem, em vez de utilizarem o computador como centralizador de informações. Assim, caso o utilizador atualize alguma informação através de um dispositivo *Apple*, o *iCloud* pode enviar esses mesmos dados atualizados automaticamente para os outros dispositivos.

3. Desenvolvimento

Este capítulo descreve o processo de desenvolvimento do protótipo. Na primeira parte do capítulo são apresentadas as ferramentas usadas para atingir os objetivos inicialmente propostos. Na restante parte, serão demonstradas partes relevantes do processo de desenvolvimento.

Após explicação do processo de desenvolvimento, alguns *screenshots* serão apresentados, em ordem para mostrar o que foi alcançado e qual o seu funcionamento.

3.1. Ferramentas, tecnologias e metodologias usadas

Para o desenvolvimento do protótipo de sistema recorreu-se à utilização de várias tecnologias, serviços e livrarias que permitiram a interligação entre subsistemas para garantirem o seu funcionamento. Esta primeira parte foca-se na exposição, explicação das ferramentas, das tecnologias e metodologias usadas durante o desenvolvimento.

3.1.1. iOS SDK

iOS é um sistema operativo para dispositivos móveis desenvolvido pela Apple. O seu nome original era *iPhone OS* (*Operating System* – Sistema Operativo), mas foi renomeado para *iOS* em 2009. O *iOS* atualmente é utilizado em dispositivos *iPhone*, *iPod touch* e *iPad* [28]. Recentemente (em Setembro-2012) foi disponibilizada a versão 6 deste Sistema Operativo.

A arquitetura do *iOS* é bastante similar à do Mac OS X, sendo que as camadas atuam como intermediárias entre o *hardware* e as aplicações. As aplicações comunicam com o *hardware* através de um conjunto de interfaces que as protegem contra a mudança

de hardware. Deste modo as aplicações podem "correr" em dispositivos, com capacidades de *hardware* diferentes [28].

A implementação das tecnologias *iOS* podem ser vistas como um conjunto de 4 camadas (*layers*): *Cocoa Touch*, *Media*, *Core Services* e *Core OS*. As camadas inferiores do sistema são serviços e tecnologias que se encontram acessíveis aos programadores de iOS, enquanto as camadas de alto nível contêm serviços e tecnologias mais sofisticadas [28]. Na Figura 17 podemos ver as camadas ordenadas verticalmente do nível mais alto até ao mais baixo.

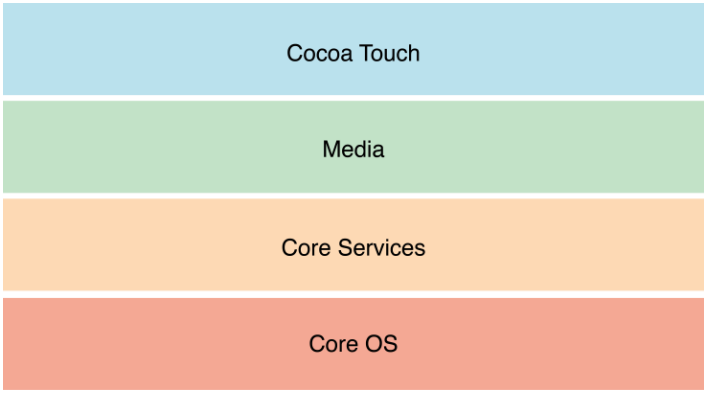

Figura 17 – Camadas do SO iOS [28]

A camada *Cocoa Touch*, escrita em *Objective-C* contém as principais *frameworks* (*UIKit* e *Foundation*) para a construção das aplicações, definindo a arquitetura básica de uma aplicação (MVC), tais como câmara, acelerómetro, localização, multi-tarefas (*multitasking*), eventos e controladores de toque múltiplos (*multi-touch*), controladores de vistas padrão (System View Controllers) [30]. A *framework Foundation* define o comportamento dos objetos básicos, sendo também essencial uma cobertura orientada ao objeto para o *Core Foundation* [29].

A camada *Media* foi implementada de modo a disponibilizar serviços multimédia, permitindo o acesso a protocolos gráficos, áudio e vídeo. As principais tecnologias desta camada são o *Core Graphics, Core Audio, Core Animation, Core Text, Core Image, Image I/O, OpenGL ES* e *GLKit* [28].

A camada *Core Services* contém os principais serviços que as aplicações utilizam. Mesmo não usando os serviços diretamente, algumas partes do sistema são construídos "em cima" destes.

Umas das caraterísticas a salientar deste nível é o *iCloud Storage*, que permite que as aplicações "coloquem" documentos do utilizador numa localização central, fornecendo o acesso a estes a partir de qualquer dispositivo iOS ou computador. Este conceito de ubiquidade usando o *iCloud*, permite, para além de uma camada de segurança, que o utilizador consiga ver ou editar os seus ficheiros através de qualquer dispositivo sem ser necessário uma sincronização ou transferência de ficheiros explicitamente. Mesmo que haja perda do dispositivo, o utilizador pode aceder aos seus documentos através deste serviço.

Quanto às *frameworks*, entre muitas, destacam-se a *Foundation*, responsável pelos tipos de dados orientados a objetos, o *Core Data*, responsável pela gestão do modelo de dados de uma aplicação *MVC*, *Core Location* responsável por fornecer localização e informação para as aplicações, *Core Media* responsável por fornecer os tipos de *media* de baixo nível utilizados pela *Foundation*, *System Configuration* responsável pelo fornecimento de interfaces de acesso que são usadas para determinar a configuração da rede de um dispositivo.

A camada *Core OS*, contém as *frameworks* e serviços de mais baixo nível, tais como gestão de *threads*, *I/O*, gestão de memória, que servem de base para a construção das restantes camadas. O uso desta camada também é usado em situações onde é preciso lidar com a segurança.

Através da figura podemos ver de forma sucinta a estrutura do iOS, dividindo-a em duas camadas, *C* e *Cocoa*. São utilizadas funções da linguagem *C* para manipular a camada *C* que representa a camada do SO. Enquanto a camada *Cocoa* sobrepõe-se à camada *C*, simplificando a programação em *iOS*, como por exemplo no uso da *Framework Foudation* string, *NSString*, em vez de usar a manipulação de strings em *C* [31].

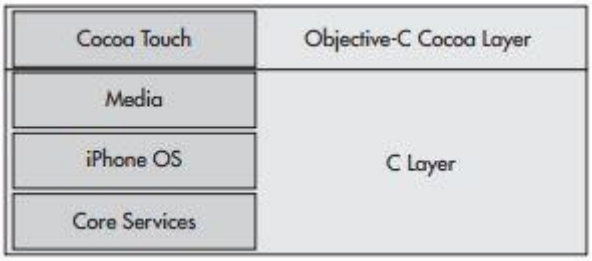

Figura 18 – Cocoa e CA como sendo as duas camadas do iOS [31]

Uma das vantagens que se deve salientar do iOS é que este dispõe de um sistema de notificações (*Apple Push Notification Service*) altamente robusto e eficiente para a propagação de informação nos dispositivos como *iPhone, iPad* e *iPod Touch*. Através deste, cada dispositivo estabelece uma ligação criptografada com o serviço que recebe notificações. Apesar de uma aplicação não se encontrar em execução, a utilização deste sistema (trabalhando em background) permite que, quando chegar uma notificação, o dispositivo avise o utilizador que a aplicação tem dados novos [35].

Notificações *Push* foram introduzidas na versão 3 do *iOS* e na versão 10.7 do *OS X*, enquanto as notificações locais (*Local notifications*) foram introduzidas na versão 4, mas não se encontram disponíveis no *OS X* [35].

Figura 19 - Sistema de notificações iOS [35]

Na Figura 19 podemos que o utilizador recebe um notificação de um evento, podendo entrar na aplicação associada à notificação para ver o detalhe do mesmo. Podem também, os utilizadores, ignorar a notificação, não ativando desta forma a aplicação.

O *iOS SDK* é composto por diversos programas e bibliotecas de código para o desenvolvimento, sendo eles:

- *Xcode*
- *Interface Builder*
- *Instruments*
- *iOS Simulator*

3.1.2. Objective-C

Objective-C é uma linguagem orientada a objetos que é basicamente uma mistura de duas linguagens, *C* e *SmallTalk*. É definido como um conjunto pequeno mas poderoso, de extensões para a linguagem padrão *ANSI-C*, fornecendo sintaxe para a definição de classes e métodos, assim como outros tipos de estruturas que promovem a extensão dinâmica de classes. Foi desenhado para fornecer capacidades de programação orientada a objetos de *C*, fazendo-o de uma forma simples e direta [33].

Qualquer programa desenvolvido em *C* pode ser compilado com um compilador de *Objetive-C*, pois esta é uma camada que assenta sobre a linguagem *C*. Classes em *Objective-C*, também podem incluir código *C* [34].

Como acontece na linguagem *C*, em *Objective-C* são definidos ficheiros de cabeçalho (*headers*) e ficheiros de código fonte (*source code*) que separam as declarações da implementação do código. Os ficheiros *.h*, considerados os ficheiros de cabeçalho contêm classes, tipos, funções e declarações de constantes. Enquanto os ficheiros *.m*, considerados como ficheiros de código fonte, podem conter código *Objective-C* como código *C*.

3.1.3. Xcode

Para desenvolver aplicações para iOS é necessário, para além de um computador *Macintosh*, o *Xcode*. Este é um *IDE* (ambiente de desenvolvimento integrado) onde se pode fazer a gestão dos projetos, edição de código, criação de executáveis, *debug* do projeto num simulador ou mesmo num dispositivo *Apple*, entre outros [28]. Na Figura 20 pode observar-se uma janela de um projeto utilizando o *Xcode*.

Figura 20 - Janela de um projeto Xcode [28]

O *Xcode* permite ao programador, após construir a sua aplicação, escolher onde a deseja executar ou para o *iOS Simulator* ou para um dispositivo. O *iOS Simulator* fornece um ambiente local para testar as aplicações e para verificar se estas se comportam conforme o desejado [36]. A Figura 21 demonstra-nos o processo de *building* (criação) até à demonstração da aplicação no *iOS Simulator*.

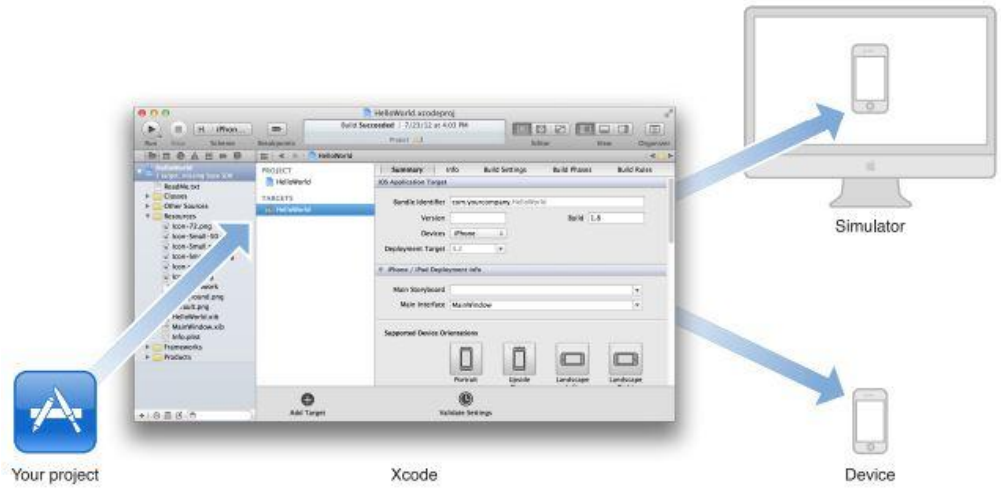

Figura 21 - Execução de um projeto através do Xcode [36]

3.1.4. Instruments

A *Apple* tem várias exigências para que as aplicações sejam publicadas na *AppStore*. Então, para que os utilizadores possam ter todas as exigências satisfeitas na altura da entrega da aplicação, a *Apple* disponibiliza a ferramenta *Instruments*, que permite analisar a performance da aplicação *iOS* enquanto esta está a ser executada no simulador ou no dispositivo. A informação desta análise, é apresentada através de uma ferramenta gráfica chamada *timeline* [36].

Através do *Instruments* o utilizador pode verificar o uso de memória da aplicação, a atividade em disco, a atividade na rede e o desempenho da parte gráfica. Podemos ver um exemplo de utilização da ferramenta *Instruments* na Figura 22.

Outra funcionalidade disponibilizada, para além do *timeline*, é a ferramenta que permite analisar o comportamento da aplicação ao longo do tempo, podendo desta forma, verificar se o seu comportamento está a melhorar ou se ainda precisa de trabalho [36].

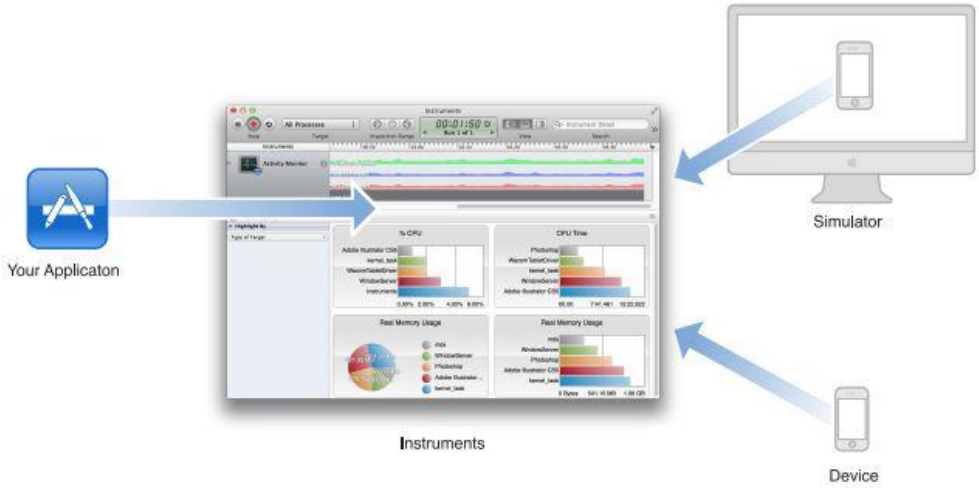

Figura 22 - Utilização da ferramenta *Instruments* [36]

3.1.5. iOS Simulator

O uso desta ferramenta serve para simular um ambiente local de teste da aplicação no *Mac*. Assim, não é necessário um dispositivo *Apple*, para testar as funcionalidades da aplicação.

Uma desvantagem do uso do *iOS Simulator*, é que este tem uma limitação a nível de suporte a algum *hardware*, pois não consegue simular, por exemplo, o uso do acelerómetro ou da câmara.

Na Figura 23 podemos observar a interface do *iOS Simulator*.

Figura 23 - Interface *iOS Simulator* [38]

3.1.6. Interface Builder

Interface Builder é um editor visual de desenho que se encontra integrado no Xcode e é usado para criar interfaces de uma forma visual para as aplicações *iOS*. Através deste pode-se criar janelas (w*indows)*, vistas (*views)*, controlos (*controls)*, menus e outros elementos através de uma livraria de objetos, bastando para isso fazer um movimento de arrastamento (ou *drag and drop*) para os inserir na aplicação [39]. Cada componente pode ser configurado, bastando para isso utilizar o *inspector* para definir as suas propriedades [30].

Na Figura 24 podemos observar o *Interface Builder* com a *dock* (área de seleção dos componentes) e o *canvas* (área de inserção das componentes).

De referir que, após a inserção dos componentes e configuração das suas propriedades, a interface é guardada num ficheiro de formato *nib*.

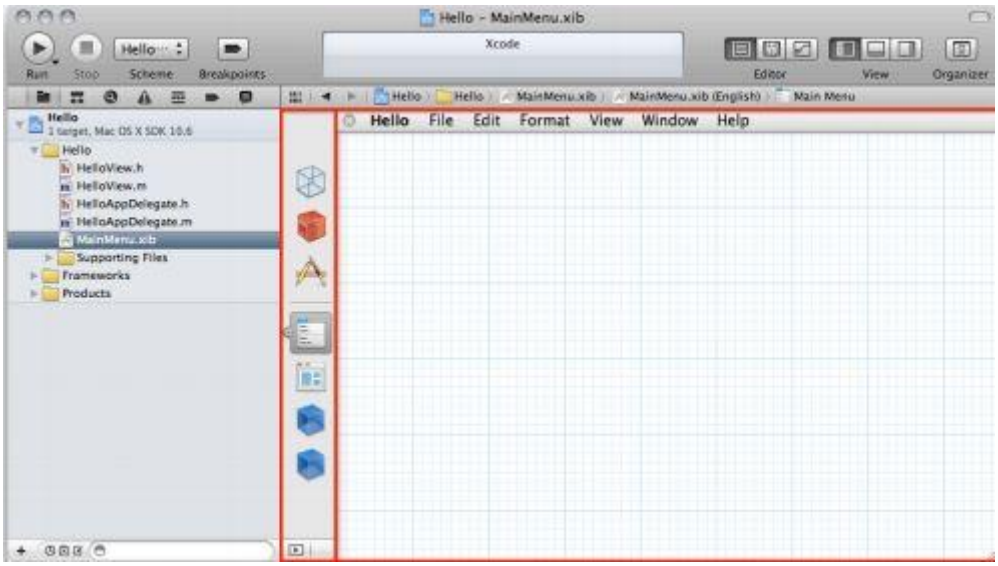

Figura 24 - *Dock* e *Canvas* do *Interface Builder* [39]

Esta ferramenta tempo como principal vantagem o fato de permitir criar interfaces de forma rápida e fácil, permitindo assim, ao utilizador que tenha uma ideia exata de como a interface irá ficar no dispositivo, isto devido à sua componente visual.

3.1.7. Model-View-Controller

MVC significa *Model-View-Controller*, um padrão de desenho que se preocupa com a arquitetura global de uma aplicação, classificando os objetos de acordo com funções gerais que são usadas nesta [43].

Programas orientados-a-objetos beneficiam deste padrão aquando a realização dos seus projetos, pois através deste, muitos dos objetos tornam-se reutilizáveis. Os programas são mais adaptáveis às mudanças dos requisitos, tornando-se extensíveis a outros programas que não são baseados em *MVC* [43].

No *iOS*, os padrões de desenho ajudam o programador a compreender a arquitetura das *frameworks* e a melhorar a estruturação do código da aplicação descrevendo as relações e hierarquias entre objetos [30].

MVC é composto por três partes que dão nome ao padrão, onde cada uma delas tem características próprias e funções bem definidas para facilitar a vida ao programador.

O ***M**odel* (modelo) mantém os dados de uma aplicação e define a lógica que manipula esses mesmos dados. Uma aplicação se for bem concebida, tem todos os seus dados importantes encapsulados em objetos do *Model*.

As ***V**iews* (vistas) representam a visualização dos dados. Estas são os elementos que são responsáveis por *renderizar* o conteúdo de uma parte particular do modelo, encaminhando para o controlador as ações do utilizador. Acedendo a partes do conteúdo do modelo, através do controlador, determinam como esses dados devem ser apresentados. Uma *View*, pode ainda, armazenar dados em *cache*, podendo posteriormente usar esses mesmos dados, de forma a melhorar o desempenho, tornando-se reutilizáveis e configuráveis, fornecendo uma maior consistência nas aplicações.

O ***C**ontroller* atua como parte intermédia entre os objetos das *Views* e os objetos do *Model*, controlando todo o fluxo de informação que passa pelo sistema.
Este é responsável por ter a certeza que as *Views* têm acesso aos objetos do modelo que pretendem mostrar [43]. Define o que deve ser acionado e para onde as informações devem ir.

A Figura 29 representa a tradicional versão do padrão *MVC*

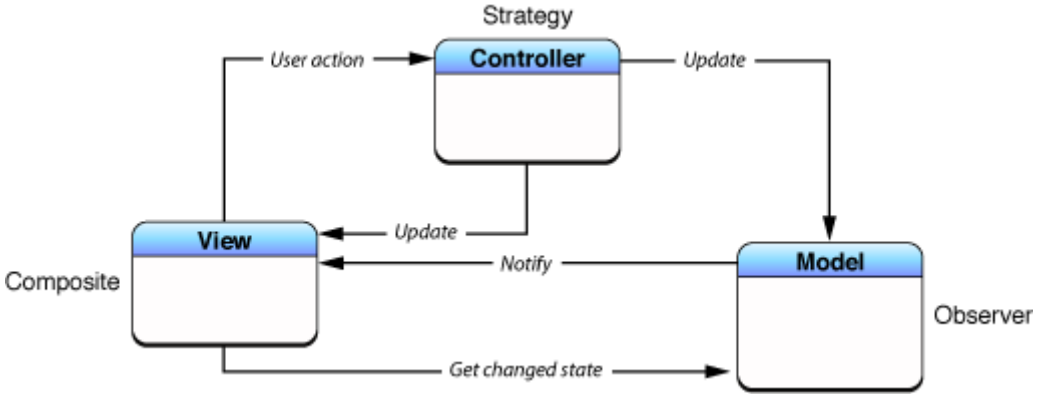

Figura 25 - *Mode-View-Controller* representação tradicional [43]

3.1.8. PHP

A construção de sistemas que trabalham em ambiente de background, implica a escolha de uma tecnologia capaz de lidar com os requisitos de sistema.

Como o uso de *web services* serão integrados neste trabalho é preciso um fornecimento de uma API para que seja possível contatar os serviços que alimentam a aplicação. Desta forma foi utilizado a linguagem *PHP* juntamente com a livraria *NuSoap* (grupo de classes *PHP* direcionado para o consumo de serviços *SOAP*), de forma a responderem aos pedidos da aplicação. A maioria destes pedidos, requerem o uso de uma camada de acesso a dados, uma vez que a informação referente a cada utilizador é guardada no *SGBD MySQL*.

PHP (acrónimo recursivo para PHP: Hypertext Preprocessor) é uma linguagem livre (*open source*) especialmente virada para o desenvolvimento de aplicações *Web* no lado do servidor, que pode ser embutido dentro de *HTML*.

3.1.9. NetBeans IDE PHP

Como foi necessário o desenvolvimento de uma *API* para alimentar a aplicação, era também necessário um *IDE* que permitisse esse mesmo desenvolvimento. Por isso foi adotado o *NetBeans IDE* que é uma ferramenta gratuita (*open source*).

O *NetBeans IDE* é reconhecido pela maior parte dos utilizadores como sendo o original *IDE* de *Java*. Porém este fornece suporte a várias linguagens de programação (*PHP, JavaFX, C/C++, Javascript*, etc) e *frameworks* [40].

Este teve início em 1996 por um grupo de estudantes da Checoslováquia, sendo que o seu nome inicial era *Xelfi* [40].

Na Figura 26 podemos observar uma visão geral da interface deste *IDE*.

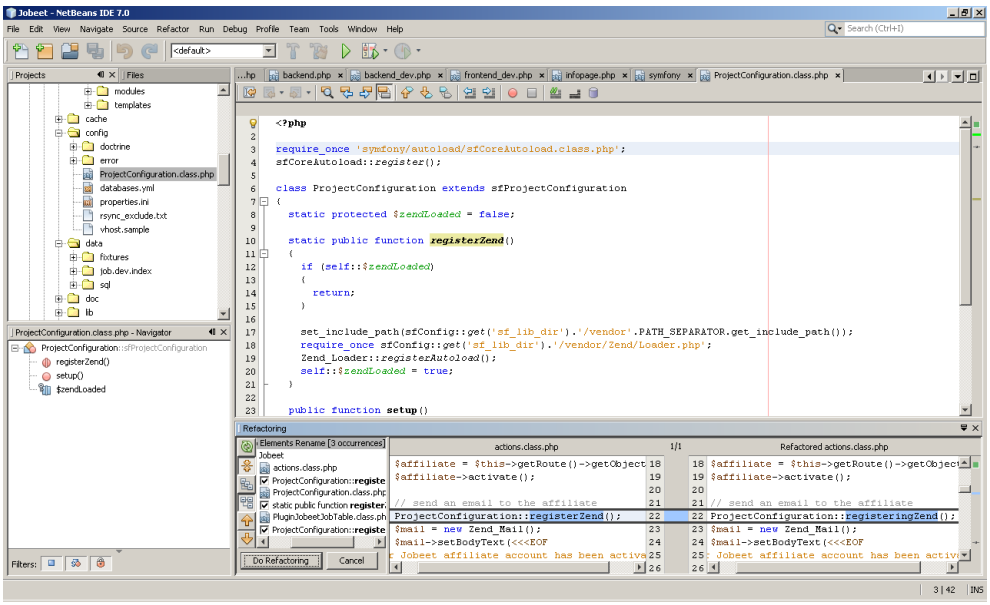

Figura 26 - Visão geral da interface do *NetBeans IDE PHP*

3.1.10. XML

Como abordamos na seção 3.1.8, o sistema faz uso de uma *API* para se alimentar. Tanto os pedidos como as respostas são enviadas via *SOAP*, que é um protocolo baseado em *XML*, respondendo com informação devidamente estruturada.

XML é uma linguagem de marcação para documentos que contém informações estruturadas. Este é um mecanismo que permite identificar as estruturas de um documento [41].

Um exemplo de um *XML*, pode ser:

```
<header>
  <file>
    <name>Documento 1</name>
    <type>PDF</type>
  </file>
</header>
```

3.1.11. Finger-Swipe (Movimento de arrastamento)

Quando os utilizadores tocam e movem os seus dedos sobre os dispositivos (ex. *iPAD*), geram eventos multi-toque, eventos de gesto e eventos de seleção. O *hardware* deste tipo de dispositivos tem suportes embutidos que permitem interpretar os gestos comuns.
Quando é efeito um movimento de arrastamento é enviado uma ação única por gesto.

O uso de efeitos visuais combinados com eventos de toque permitem criar aplicações interativas permitindo ao utilizador manipular objetos do ecrã com os seus próprios dedos, tornando a interação humano-computador bastante interessante.

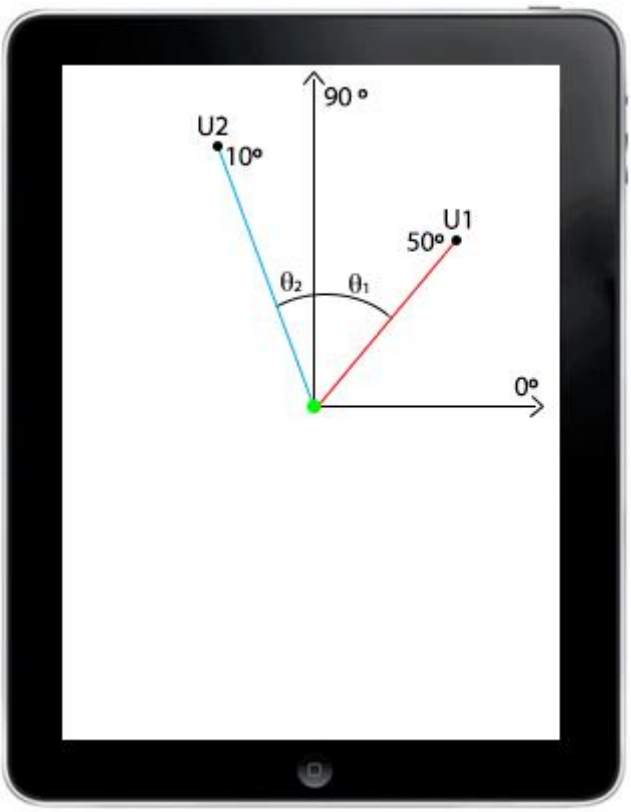

Figura 27 – *Finger swipe* com o radar

Para podermos definir qual o utilizador a ser intercetado aquando do movimento *finger swipe*, necessitamos de efetuar alguns cálculos.
Assumindo o ponto verde, como sendo o ponto referencial (0, 0), mediante o movimento efetuado pelo utilizador, é obtido o ângulo a partir da conjugação entre o

vetor de deslocação do *finger-swipe* e o vetor efetuado pelos pontos (0,0) e posição de cada utilizador. A partir desse ângulo gerado, é verificado se o mesmo está dentro do máximo definido pelo protótipo (que é 10º) e caso esteja, o pedido de transferência pode ser efetuada. Caso contrário, será ignorada a ação.
Caso existam dois ou mais utilizadores com o mesmo ângulo de diferença, então é a menor distância entre o ponto de cada utilizador e o ponto referencial que dita qual o utilizador escolhido para a transferência.

3.1.12. Realidade Aumentada

A Realidade Aumentada (*AR*) é uma área que tem vindo a evoluir ao longo do tempo, sendo parte integrante na investigação da realidade virtual, que por sua vez mistura o mundo real com o mundo virtual.

A *AR* possui algumas configurações, sendo que uma delas funciona através do reconhecimento de padrões. Aqui, o *software* processa uma imagem capturada por uma câmara e identifica o posicionamento do padrão, para posteriormente ilustrar um objeto virtual com base nesse mesmo posicionamento.

No protótipo desenvolvido, estamos a colocar um ponto no espaço *3D*, através dos valores de longitude, latitude e altitude, não estando à procura de nenhum padrão. Sabendo onde este (ponto) se encontra e através do nosso sistema ilustramos visualmente o utilizador.

No âmbito do nosso projeto, recorremos a técnicas de realidade aumentada, onde o sistema mistura o sinal de vídeo com o radar responsável pela representação geográfica de utilizadores. Na eventualidade de, algum utilizador estar no raio visual do dispositivo, não só esse utilizador será visto na captura de vídeo, como também representado no radar de georreferenciação. Por sua vez, o executar de movimentos de *finger swipe* sobre o ecrã do dispositivo tem em conta o centro do radar e a

localização do utilizador nesse mesmo radar. Assim e mais uma vez, misturamos o mundo real com o mundo virtual.

3.1.13. Core Location

O *Core Location* usa três diferentes métodos para determinar a localização. Estes são o *Cell Tower Triangulation* (*Cell-ID* abordado em 2.3.1), *Wi-Fi location* e *GPS*.

A classe *CLLocationManager* define a interface para configurar a entrega de eventos de localização e a posição relacionados com o dispositivo. É usada uma instância da classe para estabelecer os parâmetros que determinam quando os eventos de localização devem ser entregues. Pode-se usar, também o objeto *location manager* para guardar a localização mais recente [46].

Dependendo do valor do parâmetro do *CLLocationManager desiredAccuracy*, este vai usar um ou mais dos três métodos para determinar a localização do dispositivo.

Primeiramente este determina uma localização com pouca precisão usando o *Cell-ID*. De seguida, tenta encontrar um ou mais *hotspots Wi-Fi*, recebendo desta forma uma correção da localização, sendo que quantos mais encontrar melhor será a localização. Por fim, se o GPS estiver disponível, a localização será praticamente a exata do dispositivo.
Ou seja, se estivermos num ambiente *outdoor*, com sinal *GPS* e com alguns *hotspots Wi-Fi* disponíveis, podemos obter uma localização mais precisa. Já em ambientes *indoor* este valor poderá ser maior.

3.2. Prova do Conceito Proposto

O sistema tenta dar uma prova de conceito, permitindo aos utilizadores procurarem por outros dispositivos (utilizadores), através de mecanismos de localização,

partilhando um dos seus documentos com os demais, através de movimentos de arrastamento no dispositivo. O sistema tem como base a linguagem *Objective-C*, para o lado do cliente e *PHP* para o lado do servidor. Este último é responsável por responder aos pedidos a partir do sistema do lado do cliente, que se encontra instalada no dispositivo do utilizador.

A solução, a partir do lado do cliente permite, para além da localização dos dispositivos e partilha de documentos, a visualização da lista de documentos associada ao utilizador, bem como os respetivos *players* para cada tipo de documento, desde *PDF Reader*, *Movie Player*, *Music Player*, *URL Player* e *Image Player*.

3.2.1. Arquitetura do Sistema

Esta seção tem como objetivo explicar a arquitetura onde assenta o protótipo *UbiShare*, de modo a dar resposta aos propósitos inicialmente abordados, tal como se pode verificar através da Figura 28.

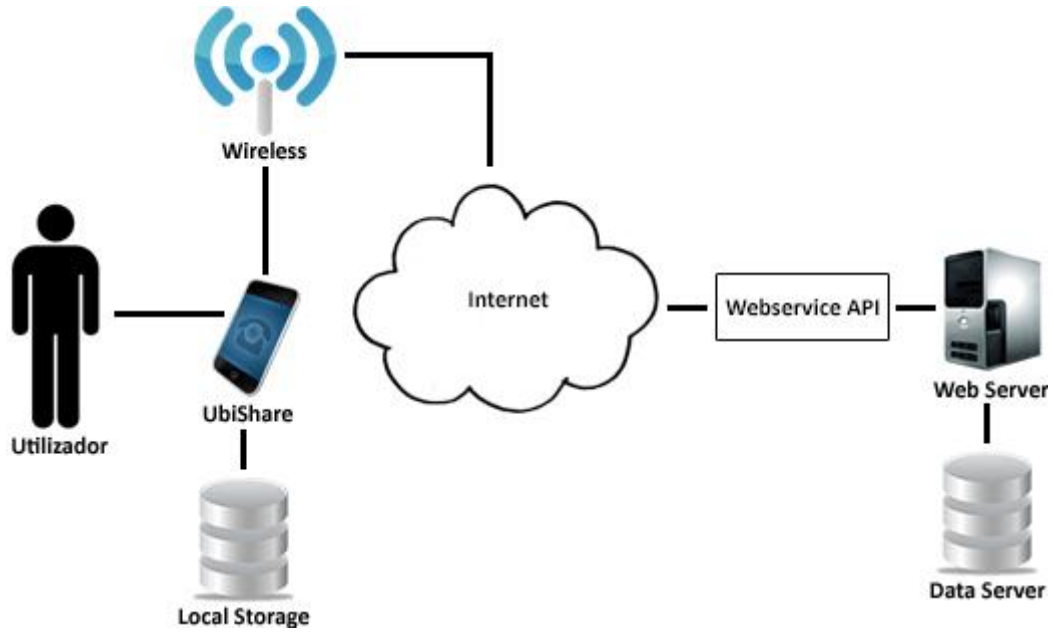

Figura 28- Arquitetura base *UbiShare*

Para um melhor entendimento daquilo que o sistema pretender oferecer, em baixo serão descritas as funcionalidades do mesmo. Como o sistema é uma prova de conceito, nem todos os recursos serão implementados.

De forma sucinta, podemos descrever o funcionamento do sistema da seguinte maneira:
O utilizador instala a *UbiShare* no seu dispositivo, tendo que, de seguida registar-se para poder ter acesso à mesma. Após preenchimento de um formulário de registo é efetuado um pedido para o lado do servidor, onde este irá responder negativamente ou positivamente ao mesmo. Caso seja positivo, quer dizer que o utilizador pode começar a usufruir das restantes funcionalidades da *UbiShare*.

Existe também, uma vista inicial de acesso à aplicação, caso o utilizador já se encontre registado, podendo ser esta chamada de *Área de Acesso*. É efetuado, também, um pedido ao servidor para verificar a veracidade dos dados introduzidos pelo utilizador, obtendo o mesmo tipo de resposta.

Após esta fase de registos e acessos, o utilizador avança para a seção de listagem, onde é construído um *SOAP request (pedido)* que é enviado ao servidor, solicitando todos os ficheiros inerentes a este, respondido através do *Webservice API (SOAP response)*, que retorna o resultado no formato *XML*. Caso este pedido falhe ou caso não hajam alterações aos ficheiros desde a última sincronização, o utilizador pode ver os seus ficheiros à mesma, pois estes são guardados no dispositivo aquando da primeira transferência. Esta operação possibilita que o utilizador, mesmo sem acesso à internet, possa aceder aos seus ficheiros, não podendo porém partilhá-los.

A partir daqui, o utilizador pode aceder ao conteúdo de cada resultado listado, utilizando para isso o *player* respetivo, sendo que o sistema é o responsável por escolher o tipo de *player* associado ao ficheiro que o utilizador pretender abrir.

Na área de listagem ou mesmo na área de cada *player*, o utilizador pode ativar a partilha do ficheiro. Aqui, é ativada a câmara do dispositivo móvel, que nos mostra no

ecrã, os pontos onde se encontram os dispositivos que também estão a utilizar a *UbiShare*. Caso haja algum ponto disponível, o utilizador pode partilhar o ficheiro, enviando um pedido ao dispositivo recetor escolhido, ficando à espera que este aceite ou rejeite a operação. Caso aceite, é iniciada a transferência do ficheiro, mediante o *cabeçalho* (que contém informação referente ao ficheiro), previamente enviado no pedido inicial. Após efetuada a transferência, o utilizador recetor ficará com esse ficheiro anexado à sua lista, podendo desta forma aceder ao seu conteúdo e fazer o mesmo processo de visualização e partilha já descrito.

Através da Figura 29 podemos ver o Diagrama de Casos de Uso.

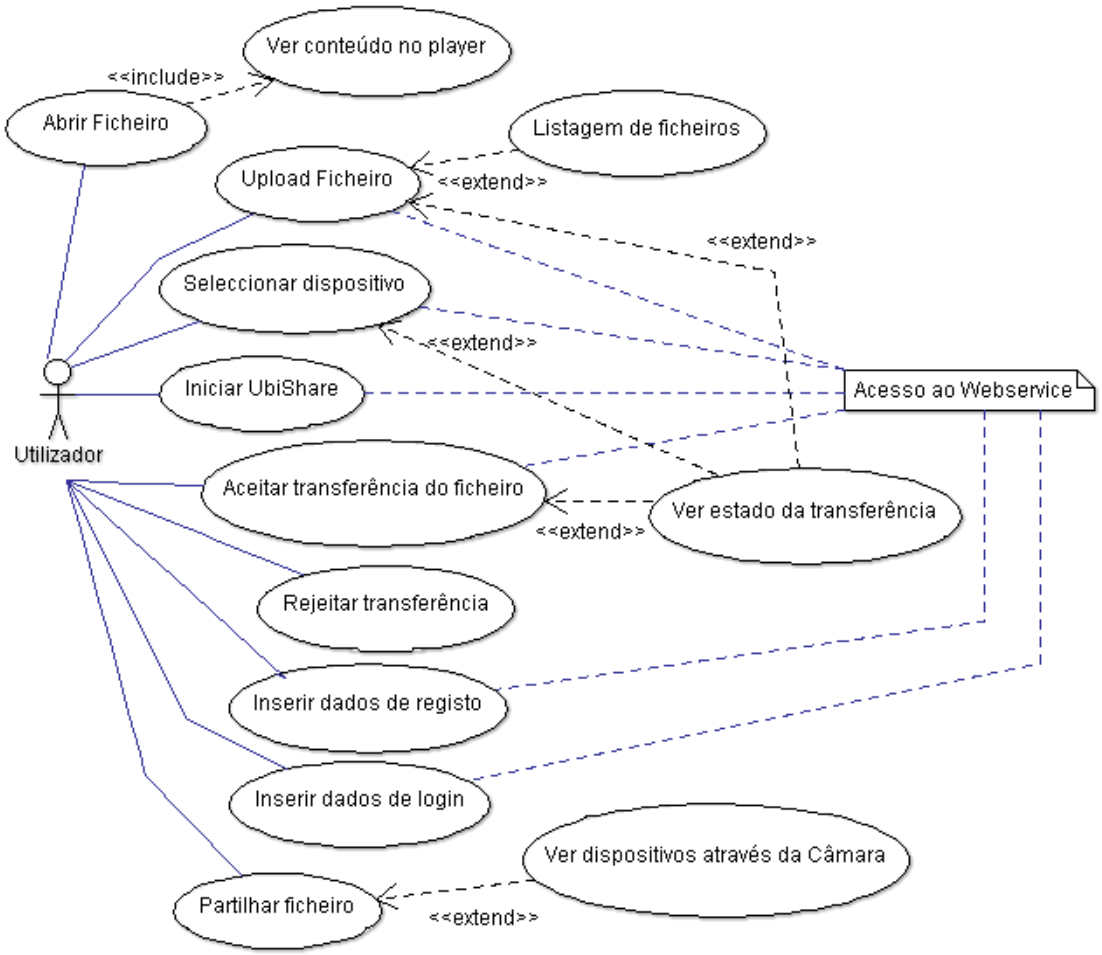

Figura 29 - Diagrama de Casos de Uso da Prova do Conceito

3.2.2. Diagrama de classes

Para a construção do protótipo foram elaborados diagramas de classes, que implementam as funcionalidades do mesmo.

3.2.2.1. Diagrama de Classes [server side]

Primeiramente, abordamos o diagrama de classes que implementa as funcionalidades do lado do servidor, onde toda a informação relativa aos utilizadores e aos ficheiros é guardada.

Para que o utilizador possa fazer uso do sistema precisa previamente de se registar, para posteriormente poder aceder aos seus ficheiros. Sendo assim, é necessária uma estrutura que suporte os dados resultantes dos pedidos efetuados ao *web service*. Na Figura 30 demonstra-se o respetivo diagrama de classes, sendo que na tabela explica-se resumidamente quais as funcionalidade das mesmas.

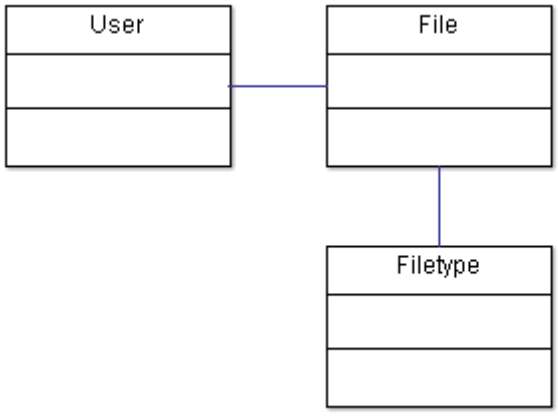

Figura 30- Diagrama de Classes [*server-side*]

Classe	**Função**
User	Contém a lista de todos os utilizadores que se registaram através da *UbiShare*,

	armazenando, também, as coordenadas atuais do utilizador e do seu dispositivo.
File	Contém a lista de todos os ficheiros adicionados pelos utilizadores através da *UbiShare* e também o objeto filetype que representa o tipo do ficheiro.
Filetype	Contém os tipos de ficheiros que se encontram disponíveis para uso na *UbiShare*.

Tabela 3 - Função das classes [*server side*]

3.2.2.2. Diagramas de Classes [UbiShare]

Finalmente aborda-se, os diagramas de classes referente às classes que suportam o funcionamento do protótipo. As tabelas, referem-se às respetivas classes e ao contributo destas.

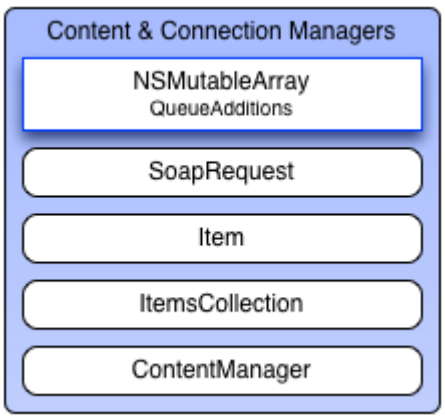

Figura 31 - Diagrama de Classes do Content e Connection Managers

Classe	Função
NSMutableArray (Queue Additions)	Complementa a classe nativa NSMutableArray, adicionando os métodos *push* e *pop*

SoapRequest	Responsável por fazer os pedidos via http ao servidor.
Item	Armazena a estrutura referente a cada documento
ItemsCollection	Dicionário de dados que armazena todos os items
ContentManager	Responsável por gerir os conteúdos do dicionário de dados, sendo também responsável por efetuar o *parsing* dos elementos recebido a cada *SoapRequest*

Tabela 4 - Funções das classes do *Content* e *Connection Managers*

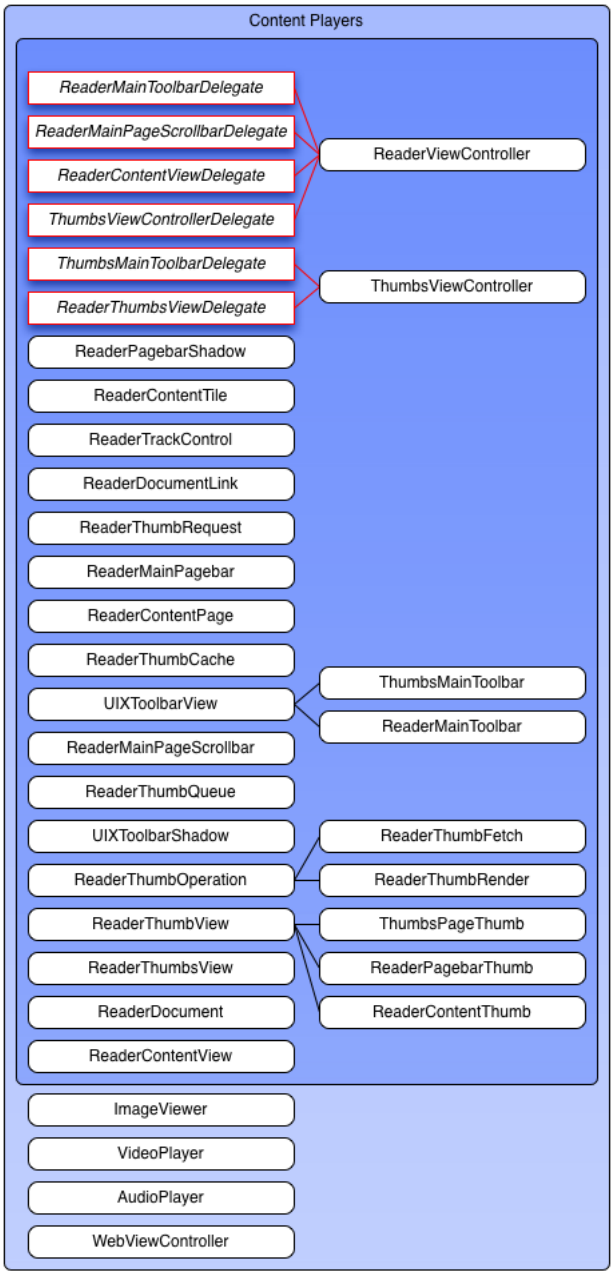

Figura 32 - Diagrama de Classes do Content Players

Classe	**Função**
ReaderMainToolbarDelegate	PDFReader: Responde aos eventos lançados pela barra de ferramentas
ReaderMainPageScrollbarDelegate	PDFReader: Responde aos eventos lançados pelo conjunto das *views* das

	páginas
ReaderContentViewDelegate	PDFReader: Responde aos eventos lançados pela *view* que contém cada página
ThumbsViewControllerDelegate	PDFReader: Responde aos eventos lançados pela *view* que contém os *thumbnails* das páginas
ThumbsMainTollbarDelegate	PDFReader: Responde aos eventos lançados pela barra de ferramentas presentes na *view* que contém os *thumbnails* das páginas.
ReaderThumbsViewDelegate	PDFReader: Responde aos eventos lançados por cada thumbnail
ReaderPageBarShadow	PDFReader: Responsável pela sombra alocada por trás de cada página
ReaderContentTitle	PDFReader: Responsável pelo título de cada documento
ReaderTrackControl	PDFReader: Gestor de eventos do PDFReader
ReaderDocumentLink	PDFReader: Responsável por estabelecer a ligação entre os *hyperlinks* presentes nos documentos
ReaderThumbRequest	PDFReader: Responsável pela geração dos *thumbnails* a partir das páginas do pdf
ReaderMainPageBar	PDFReader: Responsável pela listagem dos *thumbnails* na barra inferior do PDFReader
ReaderContentPage	PDFReader: Responsável pelo conteúdo de cada página
ReaderThumbCache	PDFReader: Armazena as imagens dos

	thumbnails para futuras utilizações
UIXToolbarShadow	PDFReader: Responsável pela sombra alocada por trás da barra de ferramentas
ReaderThumbOperation	PDFReader: Responsável por criar o *thumbnail*
ReaderThumbView	PDFReader: *View* responsável pela apresentação do *thumbnail* na grelha de *thumbnails*
ReaderThumbsView	PDFReader: *View* responsável pela listagem em grelha de todos os *thumbnails*
ReaderDocument	PDFReader: *Container* de todos slides do documento
ThumbsMainToolbar	PDFReader: *View* responsável pela listagem de *thumbnails* na view principal
ReaderThumbRender	PDFReader: *View* responsável pela renderização dos *thumbnails*
ReaderViewController	PDFReader: *View* principal do *PDFReader* agregadora de todas as outras classes
ThumbsViewController	PDFReader: View principal do *thumbnails* agregadora de todas as outras classes de *thumbnails*
ImageViewer	Responsável pela visualização de imagens
VideoPlayer	Responsável pela visualização de vídeos
AudioPlayer	Responsável pela reprodução de áudio
WebViewController	Responsável pela visualização de uma página web

Tabela 5 - Funções das classes do *Content Players*

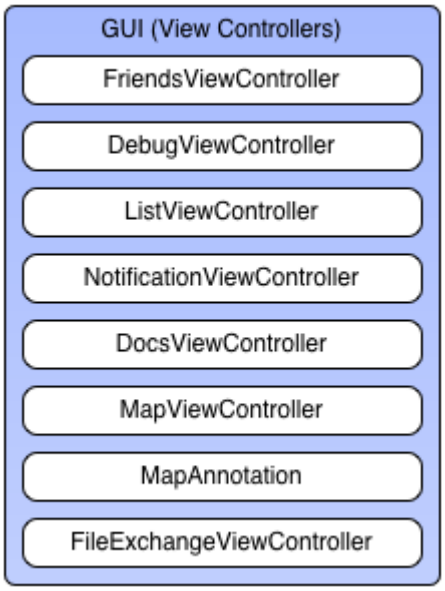

Figura 33 - Diagrama de Classes do GUI

Classe	Função
FriendsViewController	Responsável pela apresentação de todos os contatos mais recorrentes
DebugViewController	Apresenta dados de teste em modo *debug*
ListViewController	Responsável pela representação de dados em tabelas
NotificationViewController	Responsável pelo lançamento de notificações e respetiva confirmação
DocsViewController	Responsável pelos todos os documentos presentes na *UbiShare*
MapViewController	Representação dos utilizadores num mapa
MapAnnotation	Item do utilizador no mapa
FileExchangeViewController	Apresenta o estado da transferência de um documento

Tabela 6 - Funções das classes do GUI

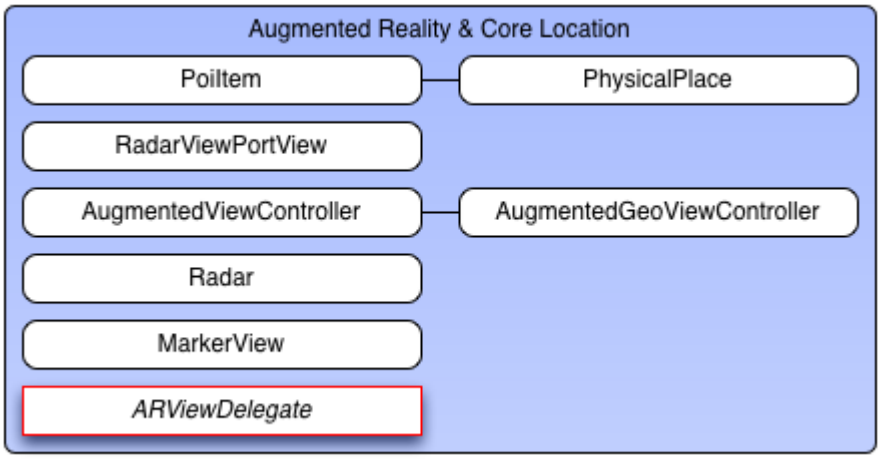

Figura 34 - Classe de Realidade Aumentada e *Core Location*

Classe	Função
PoiItem	Contém os detalhes acerca do utilizador
RadarViewPortView	Representação da região visível no radar
AugmentedViewController	Responsável pela *View* principal em realidade aumentada, misturando o radar com a captura de vídeo, a partir da câmara do dispositivo
Radar	Mecanismo de radar que representa todos os utilizadores presentes num determinado raio
MarkerView	Representação do utilizador no radar
ARViewDelegate	Responde aos eventos lançados na *view* principal de realidade aumentada
PhysicalPlace	Contém informação geográfica acerca de cada item
AugmentedGeoViewController	Camada que assenta sobre a classe principal da realidade aumentada, adicionando a noção de georreferenciação através do CoreLocation

Tabela 7 - Funções das classes da Realidade Aumentada e *Core Location*

3.2.3. UbiShare

UbiShare foi o nome dado ao protótipo que será instalado no dispositivo do utilizador. De referir, também, que o dispositivo utilizado para os testes do protótipo foi o *iPAD*. Este possui um modelo de dados que armazena localmente (*local storage*) os ficheiros que o utilizador enviou para a *Cloud* e os que lhe foram partilhados por outros utilizadores.

A primeira abordagem que o utilizador tem com o protótipo é aquela que se pode visualizar na figura.

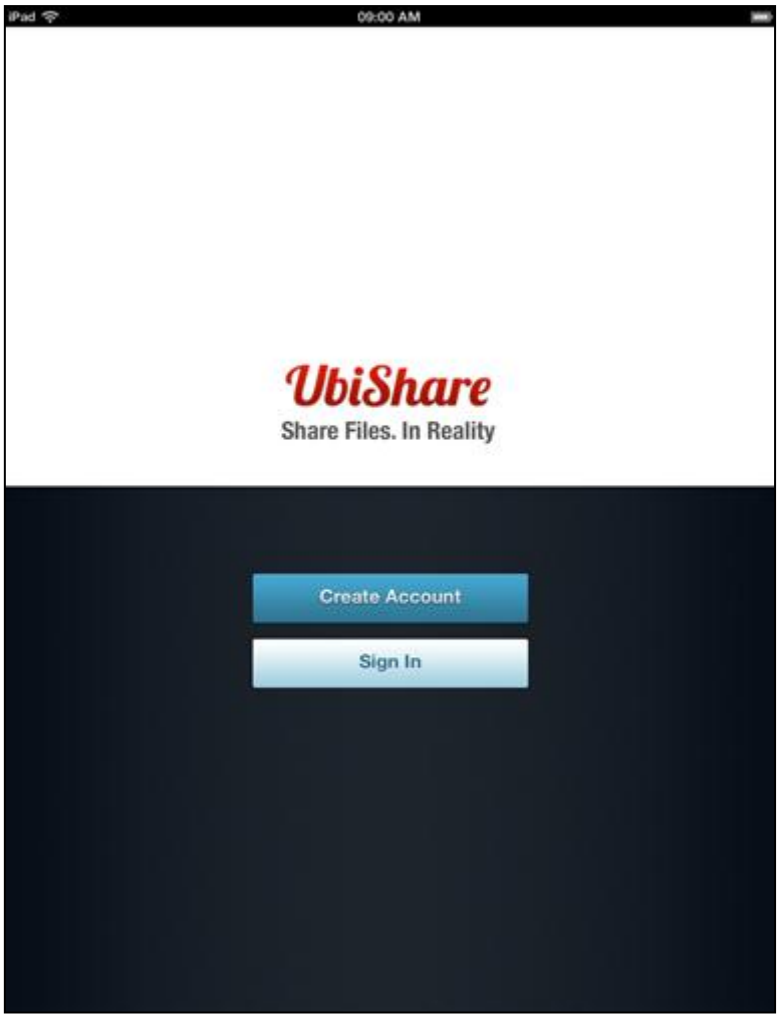

Figura 35 - Ecrã Inicial do protótipo.

Aqui o utilizador pode, caso já esteja registado, proceder ao *login* no sistema ou então criar uma nova conta. Neste primeiro contato não há qualquer tipo de pedido ao servidor.

Ao selecionar a opção de criação de nova conta, o utilizador irá ser convidado a preencher alguns campos, como se pode ver na Figura 36.

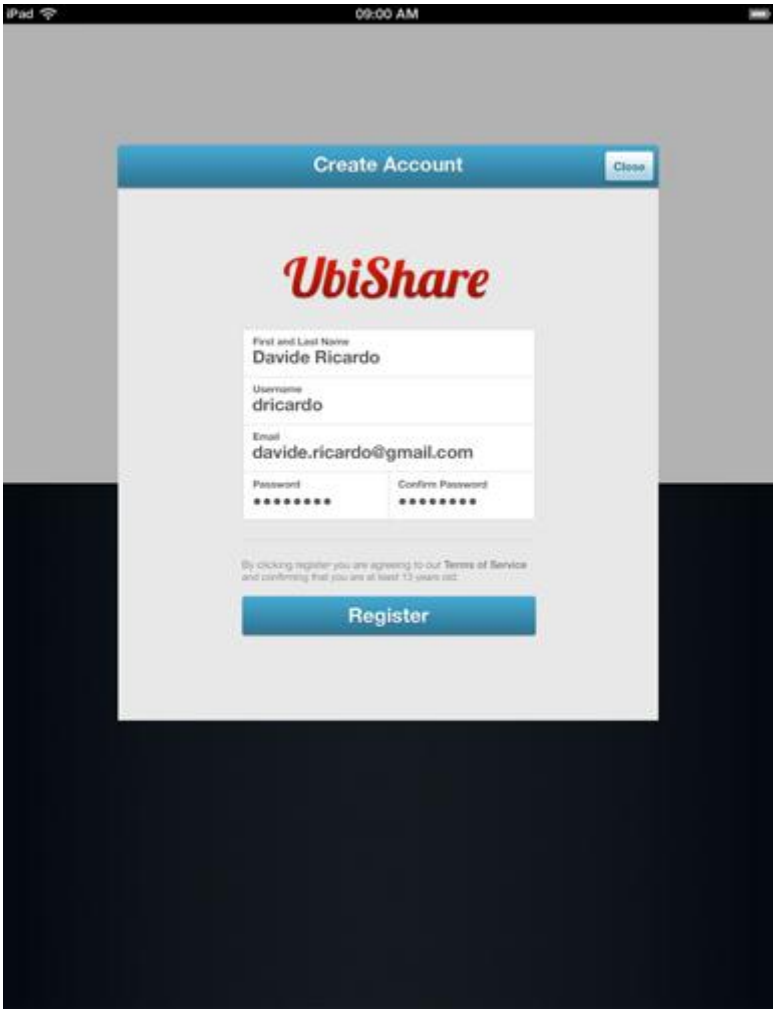

Figura 36 - Área de registo

Após o preenchimento dos campos, é pedido ao servidor a criação do novo utilizador utilizando o método *registerUser*, passando como parâmetros todos os dados solicitados no formulário. Neste pedido, e por uma questão de segurança, o campo *password* é encriptado utilizando encriptação *md5*. O resultado desta operação é um documento no formato *XML*, que contém ou o novo *ID* (identificação) do utilizador ou

o valor -1, negando o pedido. Todos os pedidos negados por parte do servidor tem como resposta o valor -1.

Caso o pedido seja positivo, é sinal que o servidor procedeu ao registo do utilizador na base dados e que, utilizando os valores recebidos cria um novo objeto *User*, definindo para cada uma das suas propriedades o valor correspondente, para posteriormente e fazendo uso da Camada de Acesso ao Objeto (*DAO*) invocar o método de inserção contido nessa mesma camada.

Se o utilizador iniciar a sua interação com o protótipo através desta seção, não necessita de passar pela fase de *login*, pois sabendo o sistema que o utilizador tem um identificador (*ID)* permite que este aceda de imediato aos seus ficheiros. Porém, o utilizador pode já estar registado, não sendo por isso necessário efetuar um novo registo, logo, para aceder aos seus conteúdos necessita de efetuar *login*, representado na Figura 37.

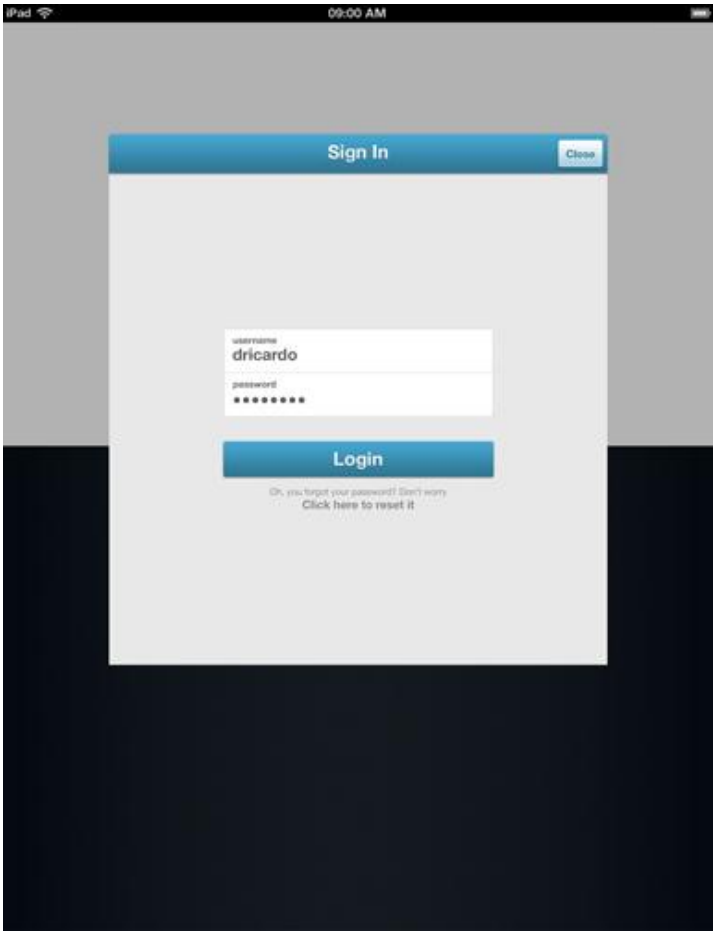

Figura 37 - Área de Login

O procedimento de *login*, funciona mais ou menos da mesma forma do de registo, com a exceção de que não irá ser feita uma adição na base dados, mas sim uma verificação da veracidade dos dados introduzidos pelo utilizador. Caso os dados introduzidos estejam corretos, a *API* responde com o *ID* do utilizador, permitindo que este siga para o próximo passo, ilustrado na Figura 38

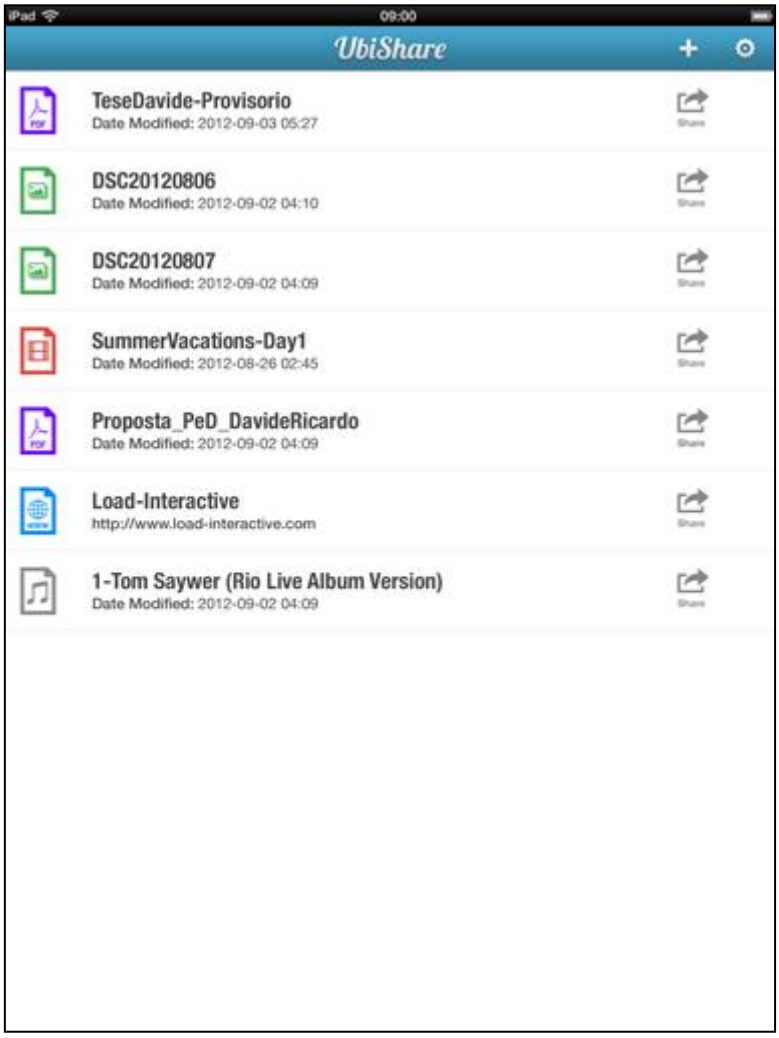

Figura 38 - *Dashboard* (Listagem de ficheiros)

Neste sistema não existe uma base dados propriamente dita, mas sim uma estrutura que guarda a informação relativa a cada ficheiro (*item)* e que é utilizada para leitura do mesmo, fazendo com que seja possível a listagem dos *items* associados ao utilizador, que foram previamente guardados num ficheiro local. Iremos abordar este assunto com detalhe mais à frente.

Através desta vista, o utilizador pode ver o conteúdo de um determinado documento, necessitando somente de o selecionar, como podemos ver na Figura 38. Neste exemplo é selecionado o documento *Proposta_PeD_DavideRicardo* que é um ficheiro do tipo *PDF*. Após feita a seleção é ativado o *player* respetivo, sendo que neste caso o *player* designa-se por *PDF Reader.* Este, tal como todos os *player*s do protótipo são nativos. Na figura podemos ver o exemplo da vista do *player* referenciado.

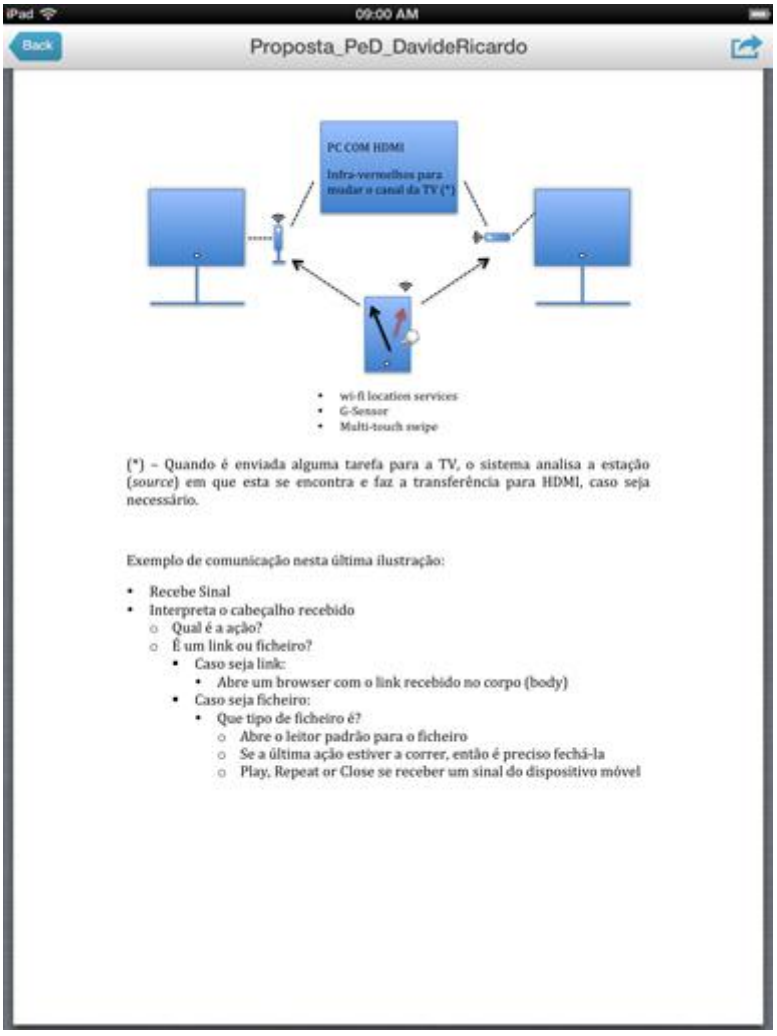

Figura 39 - Exemplo do player *PDF Reader*

Demostraremos os vários tipos de *players* disponibilizados neste protótipo antes de abordarmos a seção de partilha de conteúdo com outros dispositivos. Assim sendo, após ilustrarmos o *player PDF Reader*, podemos ver na Figura 40, o *player* de vídeo.

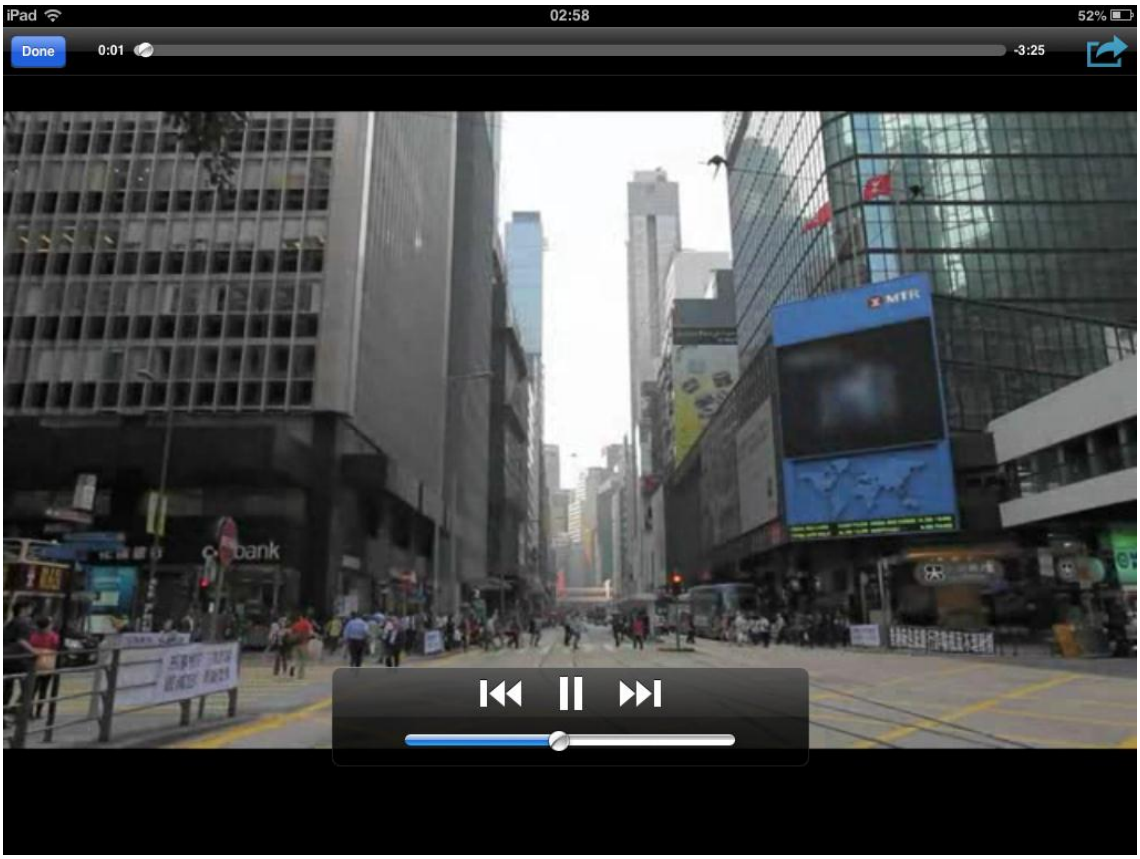

Figura 40 - Exemplo do *player* de vídeo

Neste *player*, tal como no *player* de música, demonstrado na próxima figura, a navegação é possível utilizando a *timeline* indicada no topo do *player*, avançando no tempo do vídeo/música. Podemos também utilizar os botões de navegação no fundo do *player*, que o permitem pausar, aumentar o volume do som e voltar ao início do vídeo/música. O último botão não tem nenhuma ação associada. Este tipo de *player* é nativo do *iOS*.

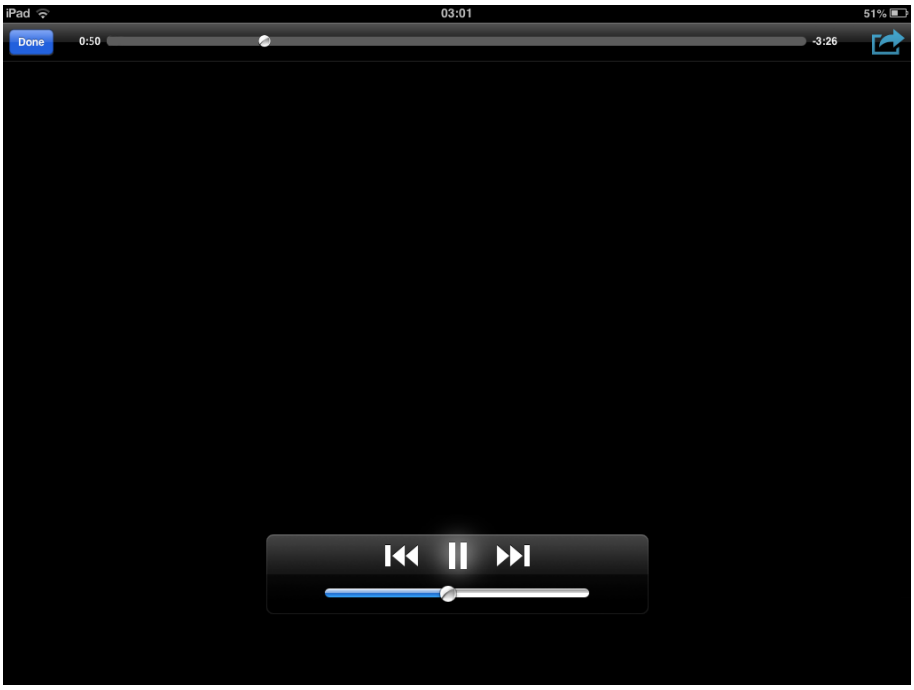

Figura 41 - Exemplo do *player* de som/música

O próximo *player*, ilustrado na Figura 42, é o de *web browser*, que permite através do *URL* abrir o *sítio* correspondente ao mesmo. Este *player* é carregado numa *UIWebView.*

Figura 42 - Exemplo do *player* WebBrowser

Por fim temos o *player* de Imagem, que nos permite abrir um determinado ficheiro do tipo imagem numa vista criada para o efeito. Na figura demonstra-se um exemplo deste *player*.

Figura 43 - Exemplo do *player* Imagem

Após a abordagem aos *players* disponibilizados neste protótipo, passamos para a abordagem ao sistema de partilha de conteúdos.

Em cada um dos *players* e mesmo na listagem inicial dos ficheiros na *Dashboard*, existe um botão de partilha, que quando acionado ativa a câmara do dispositivo móvel, de modo a detetar os utilizadores que se encontram ativos na *UbiShare.* Estes utilizadores

estão constantemente a enviar/atualizar a sua localização (latitude e longitude) para o sistema, invocando o método *UpdateCoordinates*.

A Figura 44 demonstra-nos os utilizadores que estão ativos (neste caso, somente um) no momento da ativação da câmara, sendo que, o envio do ficheiro aberto é possível quando fazemos o movimento de arrastamento para o utilizador pretendido.

Este movimento só é possível para os utilizadores que se encontram no alcance do dispositivo, sendo que esse alcance depende da distância ao dispositivo e também de estar dentro do campo de visão gerado pela orientação do dispositivo.

Figura 44 - Câmara com radar

Na Figura 44, a imagem que capturamos na câmara é representada pela seção com maior opacidade.

Após o pedido de envio, é apresentada uma notificação no dispositivo do recetor, conforme podemos verificar na Figura 45, que indica que um determinado utilizador pretende partilhar um novo ficheiro (no exemplo "Proposta_PeD_DavideRicardo.pdf"), com este. Para que seja possível a transferência, o recetor, necessita de permitir o processo através do botão *Accept*.

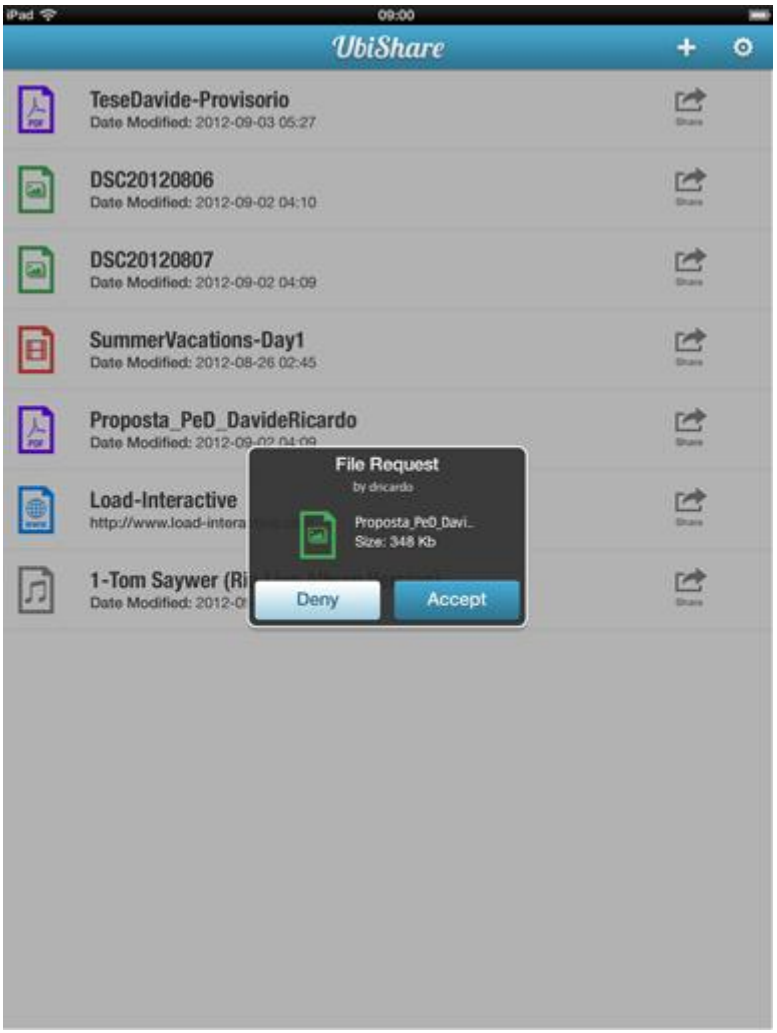

Figura 45 - Exemplo de pedido de envio de um ficheiro

Quando existe um pedido de transferência de um documento entre dois dispositivos, o recetor recebe do servidor a estrutura referente ao documento (nome do ficheiro,

tipo, link para download, etc.). São então executados uma série de passos para que a transferência seja executada.

Através da função get_downloaded_file_link é procurado na estrutura recebida o campo referente ao caminho do ficheiro e é iniciado o *download* de forma assíncrona. Optamos por executar o *download* de forma assíncrona, pois a chamada para o download é feita na *thread* principal e a transferência poderá levar algum tempo até ser concluída. Desta forma, o utilizador poderá não só continuar a usar a aplicação, não ficando a mesma bloqueada. Esta operação é iniciada através do seguinte método:

NSURLRequest *dataRequest = [NSURLRequest requestWithURL:document_url cachePolicy:NSURLRequestReloadIgnoringLocalCacheData timeoutInterval:60].

Esta ligação obriga ao estabelecimento de uma classe responsável pelo tratamento dos eventos lançados pela *NSURLRequest*. É então definido o comportamento para 4 *callbacks* que são lançadas pelo URLRequest: na primeira (*connection:didReceiveResponse*) é alocada memória para a receção do documento, criando um *container* (*NSMutableData*) vazio. Sempre que a *callback connection:didReceiveData* é chamada, são armazenados em memória os pacotes de dados recebidos. Por fim, ao terminar a transferência, é chamada a *callback connectionDidFinishLoading*, onde se cria o ficheiro com o conteúdo que tem vindo a ser armazenado em memória (no *NSMutableData*) no método *connection:didReceiveData*. É assim solicitado o caminho para armazenamento dos documentos dentro da aplicação, através do método *NSSearchPathForDirectoriesInDomains*, criando assim o documento nesse mesmo caminho. É também definida a *callback connection:didFailWithError*, responsável pelo tratamento de erros ocorridos na transferência dos dados.

Finalizada a criação do ficheiro parte-se para o processo de gravação da informação na estrutura de dados.

Mediante o recebido através da chamada SOAP (abordado em 3.2.1.), é armazenado numa *NSString* a informação e posteriormente feito o processo de *parsing* dos elementos, fazendo uso da função *parse_headers* e consequentemente da classe *NSXMLParser*, que é responsável pelo tratamento da estrutura *XML*. O que este método faz é pesquisar todos os elementos que são passíveis de serem guardados no dicionário de dados da aplicação (*ItemsCollection*). Este dicionário de dados é um *NSMutableDictionary* que contém toda a informação relativa a cada ficheiro (*Item*).

Por fim, após o *parsing* da informação contida na *NSString*, é invocada a função *write_to_cache* que guarda num ficheiro aquilo que está no dicionário de dados a cada momento, ou seja, todos os *Items* presentes em *ItemsCollection*.

Posto isto, e dada a permissão de transferência do ficheiro, é apresentada uma vista (*view)* a cada utilizador (emissor e recetor), que indica o estado do processo de envio. A Figura 46 ilustra o processo de envio de um ficheiro do lado do emissor, e a Figura 47 ilustra o processo de recebimento de um ficheiro.

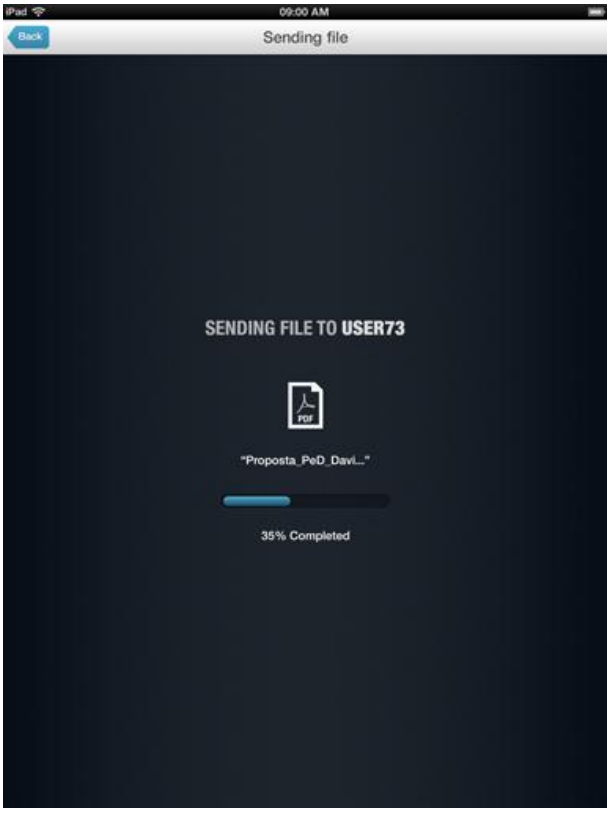

Figura 46 - Vista de transferência de ficheiro do lado do emissor

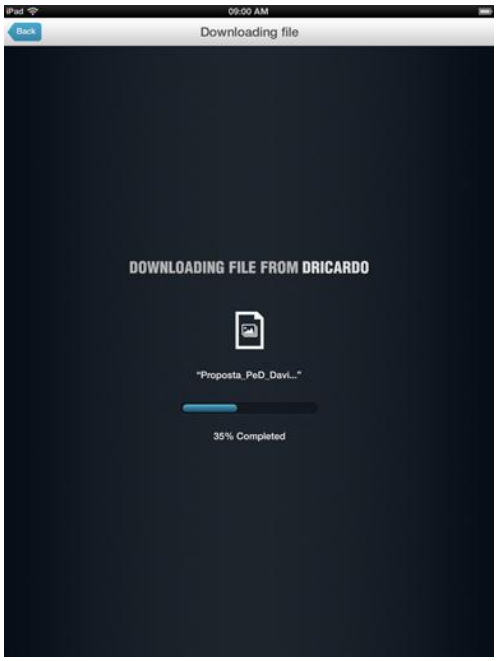

Figura 47 - Vista de transferência de ficheiro do lado do recetor

Existe ainda uma última seção para fechar o protótipo, conforme podemos ver na Figura 48. Aqui, quando é solicitado o fecho do protótipo, é invocado um método que irá guardar na estrutura de dados, tudo o que foi alocado no dicionário de dados (*ItemsCollection)*.

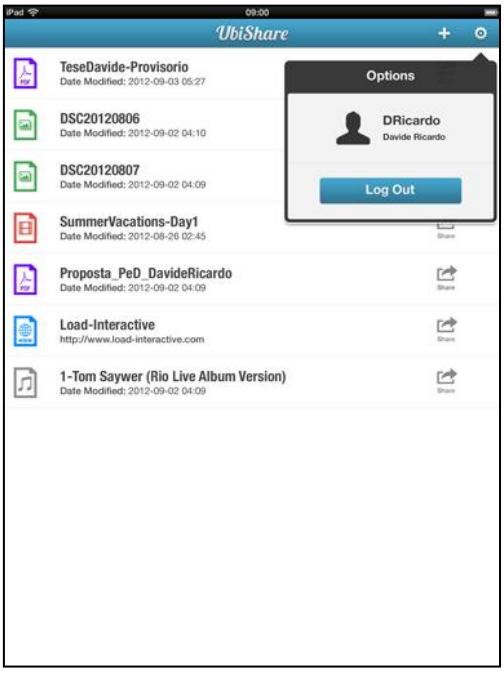

Figura 48 - Vista para fecho do protótipo

4. Conclusões e Trabalho Futuro

Neste capítulo apresentam-se as principais conclusões obtidas durante o estudo e implementação realizadas, com o intuito de responder aos propósitos inicialmente abordados. Para terminar, serão apresentadas algumas ideias para expandir futuramente o trabalho desenvolvido.

4.1. Conclusões

O uso de dispositivos móveis e dos vários tipos de serviços a eles direcionados está em crescente evolução e foi importante analisar qual o rumo que estes estão a ter e a importância que têm no quotidiano das pessoas.

A primeira fase da dissertação prendeu-se pela perceção, análise e estudo (não por esta ordem necessariamente) daquilo que eram os requisitos inicialmente propostos, tendo encontrado abordagens interessantes e por si só inovadoras. Este mesmo estudo permitiu que fossem analisadas tecnologias e mecanismos associados aos dispositivos móveis que solidificaram conhecimentos nesta área. Percebeu-se também que, estes tipos de estudos são, ainda, muito pouco abordados, tornando desta forma a pesquisa e o trabalho ainda mais aliciantes.

O fato dos dispositivos móveis terem evoluído e terem sido equipados com câmaras digitais, sensores de posição e orientação, tornou possível o desenvolvimento do protótipo, que permite a visualização de informação georreferenciada.

Após os avanços efetuados no âmbito desta dissertação, e face à necessidade de estarmos também fora dos ambientes *indoor*, foi utilizado um método que nos permite a localização em ambientes *outdoor* permitindo-nos assim, abranger um maior leque de opções (utilizadores).

Foram encontradas dificuldades no que ao desfasamento do movimento *finger swipe* diz respeito, nomeadamente na tentativa de conseguir encontrar a forma ideal para atingir o utilizador pretendido.
Outra dificuldade que deve ser mencionada foi o fato de nem sempre a triangulação ser suficientemente eficaz para procedermos à localização do dispositivo móvel.

Contudo, as metas inicialmente traçadas foram atingidas com sucesso, como tivemos oportunidade de fomentar através da *Prova do Conceito Proposto*.

4.2. Trabalho Futuro

Como trabalho futuro podem ser melhorados alguns aspetos da interface do protótipo, dando como exemplo a área de visualização dos utilizadores através da câmara.

Uma funcionalidade que não deve ser deixada de parte é permitir que, mesmo que o utilizador não esteja a utilizar o protótipo, receba notificações de novos pedidos de partilha através dos *notification services* do *iOS* ou mesmo via email.

Para incrementar a ubiquidade deste sistema poderia ser desenvolvido uma nova portabilidade do mesmo para dispositivos que não sejam *iOS*.

Uma outra funcionalidade que traria qualidade ao protótipo seria apresentar um mapa (aspeto *GoogleMaps*), que fazendo uso da ferramenta *zoom* conseguisse chegar/visualizar a utilizadores que se encontram em longa distância, permitindo o envio de documentos não por movimentos *finger-swipe*, mas sim pela seleção direta e manual do utilizador.

5. Referências Bibliográficas

[1] - Weiser, M. - “The Computer for the 21st Century”, Scientific American, 265(3):94-104, September 1991.

[2] - Weiser, M., Brown, J.S. - The coming age of calm technology.

[3] - Shafer S., Krumm J., Brumitt B., Meyers B., Czerwinski M., Robbins D. - The New - EasyLiving Project at Microsoft Research. [http://research.microsoft.com/apps/pubs/default.aspx?id=68393]

[4] – Domingues F. [http://www.hardware.com.br/artigos/computacao-ubiqua/]

[5] - Román M., Hess C., Cerqueira R., Ranganathan A., Campbell R.H., Nahrstedt K - Gaia: A Middleware Infrastructure to Enable Active Spaces

[6] – Román M., Hess C., Cerqueira R., Ranganathan A., Campbell R.H., Nahrstedt K. - Gaia: A Middleware Platform for Active Spaces

[7] – Johanson B., Fox A., Winograd T. - The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms

[8] – Streitz N.A., Tandler P., Müller-Tomfelde C., Konomi S. - Roomware: Toward the Next Generation of Human-Computer Interaction Based on an Integrated Design of Real and Virtual Worlds

[9] – Streitz N.A., Geißler J., Holmer T. - Roomware for Cooperative Buildings: Integrated Design of Architectural Spaces and Information Spaces

[10] – Zhong Y., Li X., Fan M., Shi Y., 'Doodle space: painting on a public display by cam-phone', (Beijing, China: ACM).

[11] – José R. – Introdução à Computação Ubíqua, Univ. Minho 2011

[12] – Bahl P., Padmanabhan V.N. – RADAR: An in-building RF-based user location and tracking system. In Proceedings of IEEE INFOCOM, volume 2, pages 775-784, March 2000.

[13] – Bahl P.; Padmanabhan V.N. – Enhancements to the RADAR user location and tracking system. San Diego: Microsoft Research, 2000. (Technical Report MSR-TR-2000)

[14] – Carvalho J. – Localização de Dispositivos Móveis em Redes Wi-Fi, UTAD-2007

[15] – Brás, L.P.M. – Desenvolvimento de sistema de localização indoor de baixo consumo, UNIV-AVEIRO 2009

[16] – Ekahau Inc. – Ekahau Positioning Engine 4.1, 2007

[17] –Youssef M., Agrawala A. – The Horus WLAN Location Determination System, April 1999

[18] – Moura A.I. – WBLS: um sistema de localização de dispositivos móveis em redes Wi-Fi, São Paulo 2007

[19] – Nunes B. A. A. – Um sistema de localização para redes Wi-Fi baseado em níveis de sinal e modelo referenciado de propagação.

[20] – Guedes E. – Estudo de técnica híbrida de localização de estações móveis baseada em TDoA e AoA, Rio de Janeiro 2003

[21] – R. Want, A. Hopper, V. Falcao, and J. Gibbons – The Active Badge Location System, Cambridge England 1992

[22] – Cisco Systems, Inc. – Wi-Fi Location-Based Services 4.1 Design Guide, May 2008

[23] – Yamasaki R., Ogino A., Tamaki T., Uta T., Matsuzawa N., Kato T. – TDOA location system for IEEE 802.11b WLAN, Japan 2005

[24] – Song L., Kotz D., Jain R., He X. – Evaluating location predictors with extensive Wi-Fi mobility data, 2004

[25] – Hazas M., Scott J., Krumm J. – Location-Aware Computing Comes of Age, 2004

[26] – http://www.youtube.com/watch?v=o4_CcNLd2iE

[27] – Software Engineering Institute – Basic About Cloud Computing, Carnegie Mellon University 2010

[28] – Apple Inc., iOS Tecnhnology Overview [http://developer.apple.com/library/ios/Documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf]

[29] – Jopia R. – Desenvolvimento de aplicativos para dispositivos movies – São Paulo, 2010

[30] – Nunes, D. C. – Arquitectura de Aplicações para Projecto de Engenharia em Plataformas Móveis, FEUP 2010

[31] – Brannan J. A. – iPhone SDK Programming, A Beginner's Guide, McGraw-Hill

[32] – Apple Inc. – Programming with Objective-C [http://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ProgrammingWithObjectiveC.pdf]

[33] – Apple Inc. – The Objective-C Programming Language [http://developer.apple.com/library/mac/documentation/cocoa/conceptual/ObjectiveC/ObjC.pdf]

[34] – Kochan S. G. – Programming in Objective-C 2.0, Second Edition, 2008

[35] – Apple Inc. – Local and Push Notification Programming Guide [http://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/RemoteNotificationsPG.pdf]

[36] – Apple Inc. – Xcode 4 User Guide [http://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode4UserGuide/Xcode4UserGuide.pdf]

[37] – Apple Inc. – Tools Workflow Guide for iOS [http://developer.apple.com/library/IOs/documentation/Xcode/Conceptual/ios_development_workflow/iphone_development.pdf]

[38] – Apple Inc., http://www.apple.com/br/business/accelerator/develop/

[39] – Apple Inc. – Xcode Quick Start Guide [https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/xcode_quick_start/Xcode_Quick_Start_Guide.pdf]

[40] – Oracle – Welcome to the NetBeans Community [http://netbeans.org/about/]

[41] – PHP Group – Introduction [http://www.php.net/]

[42] – O'Reilly Media Inc. – A Technical Introduction to XML [http://www.xml.com/]

[43] – Apple Inc. – Concepts in Objective-C Programming [http://developer.apple.com/library/ios/documentation/general/Conceptual/CocoaEncyclopedia/CocoaEncyclopedia.pdf]

[44] – Garlan, David; Siewiorek, Daniel P.; Smailagic, Asim; and Steenkiste, Peter – Project Aura: Toward Distraction-Free Pervasive Computing, 2002

[45] – Crijns T.K.A. – Augmented Reality for Indoor Applications on mobile devices, Eindhoven University of Technology, 2012

[46] – Apple Inc. – Core Location Framework Reference [http://developer.apple.com/library/mac/documentation/CoreLocation/Reference/CoreLocation_Framework/CoreLocation_Framework.pdf]

[47] – Julius Oklamcak – PDF Reader Core for iOS, [http://www.vfr.org]

6. Anexo

Nesta seção são abordados alguns blocos de código descrevendo o seu funcionamento.

6.1. *Métodos WebService*

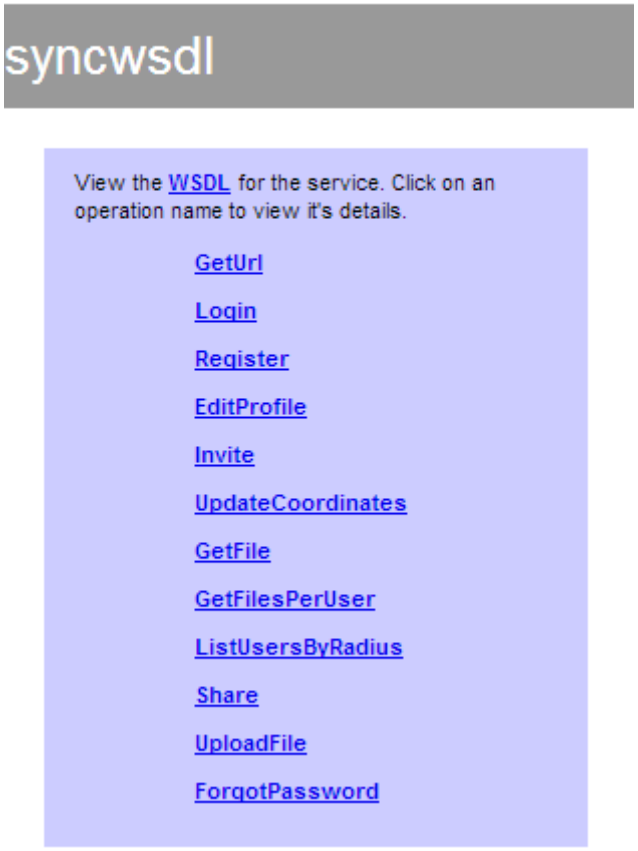

Figura 49 - Métodos WebService

A figura mostra-nos os métodos disponíveis do lado do servidor.

- GetUrl – responsável por informar qual é o link onde está alojado o webservice.
- Login – responsável por informar se os dados enviados correspondem a algum utilizador registado na base de dados.
- Register – responsável por registar um novo utilizador, ou rejeitar caso os dados enviados coincidam com algum já registado.
- EditProfile – responsável por atualizar os dados do utilizador.

- Invite – responsável por adicionar uma nova recomendação da aplicação a possíveis interessados. Aqui é enviado um email para o utilizador convidado.
- UpdateCoordinates – responsável por atualizar as coordenadas referentes à localização atual do utilizador.
- GetFile – responsável por enviar a informação relativa a um determinado *item*. Este método é invocado quando um utilizador aceita o pedido de partilha de um item de outro utilizador.
- GetFilesPerUser – responsável por devolver a lista de *items* associados ao utilizador existentes na base de dados.
- ListUserByRadius – responsável por devolver a lista de utilizadores que se encontram no raio de ação do utilizador. É informado a latitude e longitude do utilizador, bem como o raio. É efetuado um cálculo (Figura 50) para determinar a distância entre um determinado utilizador e o utilizador que está a solicitar o serviço e caso este esteja dentro do raio solicitado, é adicionado à lista.
- Share – responsável por enviar o pedido de partilha de um *item* a um determinado utilizador.
- UploadFile – responsável por enviar um ficheiro para o servidor e associa-lo a um determinado utilizador.
- ForgotPassword – responsável por ajudar o utilizador a recuperar os seus dados de acesso ao protótipo

Algumas destas funções apesar de já se encontrarem desenvolvidas, ainda não estão a ser usadas pelo protótipo.

```php
function distance($latUser1, $lonUser1, $latUser2, $lonUser2, $unit) {
    $theta= $lonUser1 - $lonUser2;
    $calc = sin(deg2rad($latUser1)) * sin(deg2rad($latUser2)) +
            cos(deg2rad($latUser1)) * cos(deg2rad($latUser2)) *
            cos(deg2rad($theta));
    $acos = acos($calc);
    $rad2deg = rad2deg($acos);
    $miles = $rad2deg * 60 * 1.1515;
    $unit = strtoupper($unit);
    if ($unit == "K") {
        return ($miles * 1.609344);
    } else if ($unit == "N") {
        return ($miles * 0.8684);
    } else {
        return $miles;
    }
}
```

Figura 50 - Cálculo da distância entre utilizadores em PHP

```xml
<header>
    <file>
        <id>
            <![CDATA[5]]>
        </id>
        <name>
            <![CDATA[Proposta_PeD_DavideRicardo]]>
        </name>
        <filename>
            <![CDATA[Proposta_PeD_DavideRicardo.pdf]]>
        </filename>
        <filetype>
            <![CDATA[pdf]]>
        </filetype>
        <dateCreated>
            <![CDATA[2012-09-02 04:09:00]]>
        </dateCreated>
        <owner>
            <![CDATA[Davide Ricardo]]>
        </owner>
        <size>
            <![CDATA[348KB]]>
        </size>
    </file>
</header>
```

Figura 51 - Exemplo de XML enviado através do servidor

6.2. ContentManager, ItemsCollection e Item

A classe *ContentManager* é a responsável por fazer o reconhecimento dos utilizadores e/ou ficheiros a serem recebidos do servidor. Ao fazer *checkForUpdates*, algo que é feito periodicamente, e no caso de haver dados a receber, é feito o parse do conteúdo

formatado em XML, por via do método *parse_headers*. Ao usar o componente *XMLParser*, teremos de definir 3 *callbacks* que são chamadas ao encontrar um elemento/tag *XML* (*didStartElement*), ao encontrar o elemento/tag de finalização XML (*didEndElement*) e ao encontrar um elemento associado a uma tag (*foundCharacters*).

Ao encontrar um elemento *file*, é criado um novo elemento *item* e dentro dessa tag *file* serão definidos todas as suas propriedades encontradas através da *callback foundCharacters*. Por fim, no fecho da tag *file*, o item é adicionado à estrutura interna *ItemsCollection*.

```
- (void)checkForUpdates
{
    if(!soap) return;

    if (![soap isConnecting])
    {
        [soap checkForUpdates];
    }

    while([soap isConnecting])
    {
        [[NSRunLoop currentRunLoop] runMode:NSDefaultRunLoopMode
                                 beforeDate:[NSDate distantFuture]];
    }

    if([soap isConnecting] || [soap hasConnectionProblems])
    {
        // TODO: set "working offline" label when working offline
    }
    else
    {
        AppDelegate *app = (AppDelegate*) [[UIApplication sharedApplication] delegate];
        NSString *webdata = app.soap_headers;

        if([webdata length] != 0)
        {
            NSLog(@"\nONLINE CONTENT:\n%@",webdata);
            [self parse_headers: webdata];
//            [self write_to_cache: webdata];
        }
    }
}
```

Figura 52 - ContentManager - checkForUpdates

```
-(void)parse_headers: (NSString *) header_contents
{
    //NSLog(@"\nONLINE CONTENT:\n%@",header_contents);

    NSData * data=[header_contents dataUsingEncoding:NSUTF8StringEncoding];

    if( xmlParser )
    {
        [xmlParser release];
        xmlParser = nil;
    }

    xmlParser = [[NSXMLParser alloc] initWithData: data];
    [xmlParser setDelegate: self];
    [xmlParser setShouldResolveExternalEntities: YES];
    [xmlParser parse];
}
```

Figura 53 - ContentManager - parse_headers

```
-(void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
 namespaceURI:(NSString *) namespaceURI qualifiedName:(NSString *)qName
   attributes: (NSDictionary *)attributeDict
{
    current_xml_element = elementName;
    //NSLog(@"xml: \n%@",elementName);

    if( [elementName isEqualToString:@"header"])
    {
        if( !collection )
        {
            collection = [[ItemsCollection alloc] init];
        }
        /*else
        {
            [collection reset_keys_registry];
        }*/
    }
    else if( [elementName isEqualToString:@"file"])
    {
        // create a new item
        current_item = [[Item alloc] init];
    }
}
```

Figura 54 - ContentManager - didStartElement

```
-(void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    if( [current_xml_element isEqualToString:@"id"] )
    {
        [current_item setItem_id: string];
    }
    else if( [current_xml_element isEqualToString:@"name"] )
    {
        [current_item setTitle: string];
    }
    else if( [current_xml_element isEqualToString:@"filename"] )
    {
        [current_item setDocument_path: string];
    }
    else if( [current_xml_element isEqualToString:@"size"] )
    {
        [current_item setSize: string];
    }
    else if( [current_xml_element isEqualToString:@"dateCreated"] )
    {
        [current_item setDate: string];
    }
    else if( [current_xml_element isEqualToString:@"filetype"] )
    {
        [current_item setDocument_type: string];
    }
    else if( [current_xml_element isEqualToString:@"owner"] )
    {
        [current_item setOwner: string];
    }

}
```

Figura 55 - ContentManager - foundCharacters

6.3. PDF Reader

Este *player* foi desenvolvido tendo como base uma versão *open-source* de autoria de Julius Oklamcak. Versão *vfr* disponível em: https://github.com/vfr/Reader ou em http://www.vfr.org

6.3.1. Reader View Controller

Esta função é a responsável pela gestão das páginas do *pdf reader*. Independente do número de páginas de um *pdf*, o sistema tem apenas preparadas 3 páginas de cada vez, por questões de performance e gestão de memória.

Assim, o que esta função faz é perceber mediante a página onde o utilizador se encontra, qual a página anterior e qual a página seguinte. Apenas a pagina atual, a

anterior e a seguinte são importantes, para que ao mudar de pagina sejam visíveis as paginas vizinhas e dar o efeito de continuidade.

A título de exemplo, quando o utilizador se encontra na página 5, as páginas carregadas são as paginas 4, 5 e 6. Ao mudar de pagina para a pagina 6, o sistema liberta memoria da pagina 4, e carrega a pagina 7, ficando a pagina 6 a partir desse momento como *pivot* (pagina central e visível). Esta função faz outras tarefas como fazer o registo das páginas navegadas (para criar uma histórico de navegação que pode ser refeito posteriormente), prepara também os *thumbnails* mediante a página atual e atualiza o número de página.

```
- (void)showDocumentPage:(NSInteger)page
{
#ifdef DEBUGX
    NSLog(@"%s", __FUNCTION__);
#endif

    isRotating = YES;

//    NSLog(@"%d != %d", page, currentPage);

    if (page != currentPage) // Only if different
    {
        // add current page to navigation history,
        //so that user can go back in pages
        if(send_pagenumber_to_navigation_history)
            [document addPageToNavigationHistory: MAX(1,currentPage)];

        send_pagenumber_to_navigation_history = YES;

        NSInteger minValue; NSInteger maxValue;
//      NSInteger maxPage = [document.pageCount integerValue];
//        NSInteger maxPage = [self document_pagecount];
        NSInteger maxPage = [document get_page_count];
        NSInteger minPage = 1;

        if ((page < minPage) || (page > maxPage)) return;

        if (maxPage <= PAGING_VIEWS) // Few pages
        {
            minValue = minPage;
            maxValue = maxPage;
        }
        else // Handle more pages
        {
            minValue = (page - 1);
            maxValue = (page + 1);

            if (minValue < minPage)
                {minValue++; maxValue++;}
            else
                if (maxValue > maxPage)
                    {minValue--; maxValue--;}
        }

        NSMutableIndexSet *newPageSet = [NSMutableIndexSet new];

        NSMutableDictionary *unusedViews = [contentViews mutableCopy];

        CGRect viewRect = CGRectZero; viewRect.size = theScrollView.bounds.size;

//        NSLog(@"NEW SETUP:");

        for (NSInteger number = minValue; number <= maxValue; number++)
        {
            NSNumber *key = [NSNumber numberWithInteger:number]; // # key

            ReaderContentView *contentView = [contentViews objectForKey:key];
```

```
    if (contentView == nil) // Create a brand new document content view
    {
        NSURL *fileURL = document.fileURL;
        NSString *phrase = document.password; // Document properties

        contentView = [[ReaderContentView alloc]
        initWithFrame:viewRect
        fileURL:fileURL page:[document translate_page: number]
        pagetag:number password:phrase];

        [theScrollView addSubview:contentView];
        [contentViews setObject:contentView forKey:key];

        contentView.message = self; [contentView release];
        [newPageSet addIndex:number];
    }
    else // Reposition the existing content view
    {
        contentView.frame = viewRect; [contentView zoomReset];

        [unusedViews removeObjectForKey:key];
    }

    [contentView go_to_offset_origin];

    viewRect.origin.x += viewRect.size.width;

}

// Remove unused views
[unusedViews enumerateKeysAndObjectsUsingBlock:
    ^(id key, id object, BOOL *stop)
    {
        [contentViews removeObjectForKey:key];

        ReaderContentView *contentView = object;

        [contentView removeFromSuperview];
    }
];
// Release unused views
[unusedViews release], unusedViews = nil;

CGFloat viewWidthX1 = viewRect.size.width;
CGFloat viewWidthX2 = (viewWidthX1 * 2.0f);

CGPoint contentOffset = CGPointZero;

if (maxPage >= PAGING_VIEWS)
{
    if (page == maxPage)
        contentOffset.x = viewWidthX2;
    else
        if (page != minPage)
            contentOffset.x = viewWidthX1;
}
else
    if (page == (PAGING_VIEWS - 1))
```

```
                contentOffset.x = viewWidthX1;

        if(CGPointEqualToPoint(theScrollView.contentOffset, contentOffset) == false)
        {
            // Update content offset
            theScrollView.contentOffset = contentOffset;
        }
        // Only if different
        if ([document.pageNumber integerValue] != page)
        {
            // Update page number
            document.pageNumber = [NSNumber numberWithInteger:page];
        }

        NSURL *fileURL = document.fileURL; NSString *phrase = document.password;
        NSString *guid = document.guid;

        if ([newPageSet containsIndex:page] == YES) // Preview visible page first
        {
            NSNumber *key = [NSNumber numberWithInteger:page]; // # key

            ReaderContentView *targetView = [contentViews objectForKey:key];

            [targetView showPageThumb:fileURL page:
             [document translate_page: page] password:phrase guid:guid];

            [newPageSet removeIndex:page]; // Remove visible page from set
        }

        // Show previews
        [newPageSet enumerateIndexesWithOptions:NSEnumerationReverse usingBlock:
            ^(NSUInteger number, BOOL *stop)
            {
                NSNumber *key = [NSNumber numberWithInteger:number]; // # key

                ReaderContentView *targetView = [contentViews objectForKey:key];

                [targetView showPageThumb:fileURL page:
                 [document translate_page: number] password:phrase guid:guid];
            }
        ];

        [newPageSet release], newPageSet = nil; // Release new page set

        [mainPagebar updatePagebar]; // Update the pagebar display

        [self updateToolbarBookmarkIcon]; // Update bookmark

        currentPage = page; // Track current page number
    }

    isRotating = NO;
}
```

Figura 56 - PDFReader - ShowDocumentPage

6.4. Player Video

Esta função é a responsável pelo *playback* de vídeo. Usando o *framework* nativo *MediaPlayer*, apenas é necessário definir o tamanho da *frame* do vídeo a carregar, o nome (e caminho) do ficheiro de vídeo, qual o tipo de controlos do vídeo *player* e define-se também a *callback* de saída (a função que é chamada ao sair de *fullscreen*

(uma vez que o vídeo é apresentado em *fullscreen*) ou no final do playback). Esta *callback* define o que fazer no final do *playback*. Por fim, apos ter o *player* configurado, é dado a instrução de play.

Para qualquer eventualidade, é feito o tratamento de qualquer exceção que ocorra aquando do *playback*, de forma a que se evite *crash* da aplicação.

```
- (void)playVideo
{
    @try
    {
        if(!playing && filepath!=nil)
        {
            NSAutoreleasePool * pool = [NSAutoreleasePool new];
//            NSLog(@"player running: %@", [filepath absoluteString]);
            player = [[MPMoviePlayerController alloc] initWithContentURL:filepath];
            player.useApplicationAudioSession=NO;
            player.view.frame=CGRectMake(0, 0, 1024, 768);
            player.shouldAutoplay = YES;
            player.scalingMode = MPMovieScalingModeAspectFit;
            player.controlStyle = MPMovieControlStyleFullscreen;
            [[NSNotificationCenter defaultCenter] addObserver:self
                    selector:@selector(exitedFullscreen:)
                    name:MPMoviePlayerPlaybackDidFinishNotification object:player];
            [self.view addSubview:player.view];

            playing = YES;
            [player play];
            [pool drain];
        }
    }
    @catch (NSException * e)
    {
        NSLog(@"Error:%@|| Reason:%@",[e description],[e reason]);
    }
    @finally
    {
    }
}
```

6.5. *Realidade Aumentada*

A função (*viewportContainsCoordinate*) é a responsável por verificar se um determinado item (utilizador) está dentro do ângulo de visão do dispositivo. Para isso, tem em consideração as coordenadas do item (inclinação e azimute) bem como as do dispositivo. Primeiramente verifica se o azimute à esquerda é positivo e se o azimute a direita é superior a 2PI, de forma a que tanto o azimute esquerdo e direito estejam dentro do range 2PI. Posteriormente verifica-se se o azimute do item esta dentro do campo de visão (planar) do dispositivo (não tendo em consideração a inclinação do

dispositivo). Posteriormente, é lida a inclinação do dispositivo e comparada com a inclinação do item (utilizador) em questão. Por fim, é feita uma conjugação entre o resultado planar e o resultado da inclinação, dando como "contido" ou "não contido" o item no campo de visão do dispositivo.

```
- (BOOL)viewportContainsCoordinate:(PoiItem *)coordinate {
    double centerAzimuth = self.centerCoordinate.azimuth;
    double leftAzimuth = centerAzimuth - VIEWPORT_WIDTH_RADIANS / 2.0;

    if (leftAzimuth < 0.0) {
        leftAzimuth = 2 * M_PI + leftAzimuth;
    }

    double rightAzimuth = centerAzimuth + VIEWPORT_WIDTH_RADIANS / 2.0;

    if (rightAzimuth > 2 * M_PI) {
        rightAzimuth = rightAzimuth - 2 * M_PI;
    }

    BOOL result = (coordinate.azimuth > leftAzimuth &&
                   coordinate.azimuth < rightAzimuth);

    if(leftAzimuth > rightAzimuth) {
        result = (coordinate.azimuth < rightAzimuth ||
                  coordinate.azimuth > leftAzimuth);
    }

    double centerInclination = self.centerCoordinate.inclination;
    double bottomInclination = centerInclination - VIEWPORT_HEIGHT_RADIANS / 2.0;
    double topInclination = centerInclination + VIEWPORT_HEIGHT_RADIANS / 2.0;

    //check the height.
    result = result && (coordinate.inclination >
            bottomInclination && coordinate.inclination < topInclination);

    //NSLog(@"coordinate: %@ result: %@", coordinate, result?@"YES":@"NO");

    return result;
}
```

Figura 57 - Realidade Aumentada - viewportContainsCoordinate