Faculdade de Engenharia da Universidade do Porto

DiABlu MailMan no Centro de Investigação e Tecnologia das Artes

Pedro Ribeiro Santos

VERSÃO 1.0

Relatório de Projecto realizado no Âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Orientador: António Augusto de Sousa (Professor Associado)

Julho de 2008

DiABlu MailMan no CITAR

Pedro Ribeiro Santos

Relatório de Projecto realizado no Âmbito do

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Jorge Alves da Silva (Professor Auxiliar)

_____

Arguente: António Ramires Fernandes
Vogal: António Augusto De Sousa (Professor Associado)

16 de Julho de 2008

Resumo

Nos tempos que correm, a arte tem um papel cada vez mais importante na vida das pessoas, e como tudo em seu redor, não pára de evoluir. A evolução na arte levou à criação da arte digital, na qual se tira proveito das novas tecnologias. Além disso apareceu também a arte interactiva, em que o público interage com a obra de arte. Embora o conceito de arte interactiva já exista há algum tempo, foram as artes digitais quem mais o explorou.

O projecto DiABlu MailMan vem expandir este conceito de arte interactiva a um novo nível. Usando uma tecnologia recente, o Bluetooth, é agora conseguida uma interacção mais dinâmica com a instalação de arte digital.

Este projecto dota as instalações de arte digital da capacidade de troca de ficheiros. Torna-se então possível usar conteúdos fornecidos pelo público para a criação de arte. Esses conteúdos podem ser imagens, vídeos, música, texto, ou qualquer outro tipo de ficheiro. Mas as melhorias introduzidas na interacção pelo uso da aplicação DiABlu MailMan não ficam por aqui, dado que esta pode ocorrer nos dois sentidos: não só o público interage com a instalação, como a instalação interage com o público. Uma instalação de arte digital recorrendo à aplicação DiABlu Bluetooth fica pois dotada de mecanismos que permitem trocar conteúdos com o público.

Este documento ilustra os principais aspectos do desenvolvimento deste projecto, realizado no âmbito da disciplina de Projecto/Dissertação do Mestrado Integrado em Engenharia Informática e de Computação da FEUP.

O desenvolvimento foi realizado em várias etapas. Na primeira etapa foi feito um estudo do estado da arte que serviu para ilustrar a falha de aplicações deste género ligadas à arte digital. Seguidamente foi feita uma análise de tecnologias, de modo a escolher as melhores para o desenvolvimento do projecto DiABlu MailMan.

A terceira etapa corresponde à especificação de requisitos e desenho da arquitectura. A quarta e última etapa do projecto corresponde à implementação, que quando completa, dá lugar à aplicação DiABlu MailMan.

Este projecto terminou com resultados satisfatórios, que culminou numa ferramenta que poderá vir a ser usada por centenas de artistas digitais.

Abstract

The arts role has become more and more important nowadays in everybody's life and, like everything around it, does not stop evolving. The evolution in art resulted in the creation of the digital arts, in which technology is used as a tool. Even though the concept of interactive art already existed, the digital arts were the ones that explored it the most.

The DiABlu MailMan Project has the objective to expand the concept of interactive art to a whole new level. Using a recent technology, the Bluetooth, it is now possible to have a more dynamic interaction with an installation of digital art.

This project makes the installations of digital art able to exchange files between them and the public. Therefore, it is now possible to use contents provided by the public in the creative process. These contents can be images, videos, music, text, or any other type of file. But the interaction that is gained by the use of the application DiABlu MailMan does not end there. This interaction can be done in both ways: not only the public interacts with the installation, but the installation itself has the capabilities to send content to the public.

This document intends to illustrate the main aspects of this project that was developed for the course of Project/Dissertation of the Integrated Masters in Informatics Engineering and Computation at FEUP.

The Project, described in this report, was separated in several stages. The first stage consisted in the study of the state of the art. The study revealed a big lack of applications dedicated to the digital arts, like this one. After this, an analysis was made to choose the best technologies for the development of the Project.

The third stage is related to the requirements specification and architecture design. Finally, the last stage of this project was the implementation. When this stage was completed, the application DiABlu MailMan was finisher. This Project ended with very satisfactory results. The DiABlu MailMan is a one of a kind application that may be used by many digital artists,

Agradecimentos

Em primeiro lugar gostaria de agradecer ao meu orientador do projecto, o professo António Augusto de Sousa por arranjar sempre tempo para mim no meio da sua agenda apertada, e por ter feito os possíveis para que este projecto decorresse da melhor forma possível.

Queria também agradecer ao Jorge Cardoso, a pessoa responsável por mim no CITAR por toda a ajuda e disponibilidade, e ainda pela confiança que depositou em mim.

A toda a equipa do CITAR com quem partilhei o espaço de trabalho e mesas de almoço: Carlos Caires, Joana Castro, Luís Sarmento, Luís Teixeira, e Pedro Baptista, agradeço pelo apoio e pelos bons momentos que me fizeram sentir integrado desde o início.

Agradeço também a todos os colegas de curso por estarem presentes nos bons e maus momentos, nas alegrias e nas tristezas, no pânico e na aceitação, na parvoíce e nos assuntos sérios, nas conversas de corredor e no silêncio dos exames e em tudo o resto porque passamos nos últimos cinco anos.

Agradeço à minha família e amigos pelo apoio que me deram não só no decorrer deste projecto, mas ao longo de toda a minha vida.

E em especial queria agradecer à minha namorada, Raquel, sem a qual, a vida não faria tanto sentido.

Conteúdos

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

A arte sempre teve um papel importante na vida dos seres humanos. Existe desde que há indícios do Homem na Terra, e atravessou todos os povos e civilizações, alterando e sendo alterada por eles. Ao longo dos tempos a arte foi alvo de uma enorme evolução, quer a nível técnico, com a criação de novas ferramentas e métodos, quer a nível filosófico, originando novas correntes artísticas.

Uma das mais recentes correntes é a arte digital, e apareceu como consequência da evolução tecnológica sentida nos últimos anos, principalmente na área da informática. Das inúmeras vantagens da arte digital, a possibilidade de criação de obras de arte dinâmicas encontra-se entre as mais importantes. A obra de arte, ou como é designado no mundo da arte digital, instalação artística, perde um pouco o estatuto de objecto estático, podendo ser algo em constante alteração. Esta nova característica foi o ponto de partida para o aparecimento da arte digital interactiva.

A interacção na arte digital pode ser feita usando qualquer dispositivo que gere dados dependendo da acção do público. Esses dispositivos podem ser de diferentes tipos, desde o rato e teclado, passado por câmaras de vídeo, até sensores de movimento ou de temperatura. O projecto DiABlu [1] apareceu para explorar uma nova interacção: dispositivos de comunicação móvel.

Na actualidade a esmagadora maioria da população possui um telemóvel. A maior parte dos dispositivos mais actuais são dotados da capacidade de comunicação por Bluetooth [2]. O protocolo de comunicação Bluetooth abre novas possibilidades de interacção. Na ferramenta DiABlu MailMan é explorada a capacidade de envio e recepção de ficheiros deste protocolo.

No processo de criação de arte digital torna-se necessário usar um conjunto de aplicações específicas para esse fim. Estas aplicações, de modo a comunicarem

entre si, tiram partido do protocolo de comunicação OSC [3] (Open Sound Control). É este protocolo que vai ser usado no projecto DiABlu MailMan para comunicar com essas aplicações.

De uma forma simplificada, o objectivo do projecto DiABlu MailMan é criar uma ferramenta capaz de dotar aplicações de criação de arte digital com capacidades de transferência de ficheiros usando o protocolo Bluetooth.

1.1 Contextualização

De modo a melhor compreender o âmbito do projecto DiABlu MailMan, é necessário compreender o contexto em que ele se insere. Por esse motivo esta secção do primeiro é dedicada a explorar os dois principais factores que influenciam este projecto. O primeiro factor, e talvez o mais importante, é a arte digital. Foi a arte digital que motivou este projecto, ou melhor dizendo, foi a arte digital que motivou o projecto DiABlu. O projecto DiABlu é o segundo factor. Pelo simples facto de o projecto DiABlu MailMan fazer parte do projecto DiABlu, torna-se necessário seguir um conjunto de regras e filosofias previamente definidas.

1.1.1 Arte Digital

A arte digital é, por definição, arte criada em formato digital recorrendo a ferramentas computacionais. Pode ser totalmente gerada por computador, como é o caso dos fractais e arte algorítmica, ou baseada numa fonte, como por exemplo uma fotografia digitalizada, ou uma imagem criada através de software de desenho vectorial usando um rato ou uma mesa digitalizadora. Embora o termo "arte digital" possa ser aplicado à arte realizada através de outros meios ou processos e só posteriormente digitalizada, é geralmente reservado para arte que tenha sido alterada de uma forma não trivial por um processo computacional, como por exemplo um programa de computador, um microcontrolador ou qualquer sistema electrónico capaz de interpretar dados de entrada e criar dados de saída. O texto digitalizado, assim como gravações de áudio e vídeo sem qualquer tipo de processamento, não são geralmente considerados arte digital por si só, mas quando inseridos num projecto mais alargado podem ser considerados como tal. Num sentido mais lato, "arte digital" é um termo aplicado à arte contemporânea que usa métodos de produção em massa de conteúdos multimédia. [1]

1.2 Projecto DiABlu

Figura 1.1 - Logótipo do Projecto DiABlu

O projecto DiABlu [2] (Digital Art's Bluetooth) começou como sendo uma aplicação que detecta dispositivos Bluetooth e que comunica com os dispositivos encontrados, através do protocolo OSC [3] (Open Sound Control), para outras aplicações que não têm a capacidade de realizar esta tarefa, de uma forma simples (como é o caso de Pure Data [4], Processing [5], Eyesweb [6], Max/MSP [7], Flash[8], etc.). O projecto tem como objectivo oferecer aos artistas digitais uma ferramenta que lhes permita adicionar interacção às suas obras a partir de dispositivos móveis. No entanto à medida que o projecto foi evoluindo, começou a fazer sentido criar outras ferramentas que permitissem interagir com dispositivos Bluetooth usando o protocolo OSC. Devido a esta evolução, o DiABlu passou a ser um projecto com o objectivo de criar um conjunto de ferramentas que forneçam capacidade de interacção às instalações de arte digital. Presentemente, o Projecto DiABlu conta já com três ferramentas que tem em comum o facto de terem uma interface OSC e comunicarem com o dispositivo com que vão interagir usando Bluetooth.

À data de escrita deste documento, o projecto DiABlu é constituído por três ferramentas, já com planos para desenvolver uma quarta num futuro próximo. As ferramentas já desenvolvidas são:

- **DiABlu Scout** – Esta ferramenta foi a primeira a ser desenvolvida no âmbito do projecto DiABlu. Apesar de ser uma ferramenta relativamente simples foi um marco importante, pois foi a primeira ferramenta a permitir o uso de Bluetooth [9] como forma de interacção em instalações de arte digital, de uma forma simplificada. O objectivo desta ferramenta à fazer uma pesquisa dos dispositivos Bluetooth existentes numa determinada área e enviar informação sobre esses mesmos dispositivos (ID, nome, classe do dispositivo, fabricante) para uma instalação de arte digital através do protocolo de comunicação OSC;
- **DiABlu LegOSC** – Com esta ferramenta é possível, a uma instalação de arte digital, controlar *robots* Lego Mindstorm NXT através de OSC. A

comunicação com o módulo de controlo do robot é feita através de Bluetooth, logo faz todo o sentido que esta ferramenta faça parte do projecto DiABlu;

- **DiABlu MailMan** – Esta é a ferramenta cujo desenvolvimento é descrito neste documento. Foi criada com o intuito de fornecer, às instalações de arte digital, capacidades de transferência de ficheiros.

O projecto DiABlu já tem em vista o desenvolvimento de uma nova ferramenta que se irá juntar ao leque já existente:

- **DiABlu SMS2OSC** – Esta será a próxima aplicação a ser desenvolvida no âmbito deste projecto. Consiste numa aplicação que permite enviar e receber mensagens SMS e, possivelmente MMS, num computador usando um telemóvel comum como *gateway*. O computador comunicará com o telemóvel através da tecnologia Bluetooth. As mensagens SMS e MMS serão posteriormente encaminhadas para outras aplicações usando o protocolo OSC.

1.3 Análise do problema

O DiABlu MailMan é a ferramenta cujo desenvolvimento é relatado neste documento. Está inserida no Projecto DiABlu e tem como objectivo base capacitar as instalações de arte digital com funcionalidades Bluetooth.

A ferramenta DiABlu MailMan veio criar uma nova forma de interacção com as instalações de arte digital, através do uso de dispositivos de comunicação móveis, como é o caso dos telemóveis. Apesar de a tecnologia Bluetooth já ter mais de uma década, só recentemente se tornou viável o seu uso, devido à crescente produção e venda de telemóveis com essa capacidade.

Esta ferramenta permite a transferência de ficheiros entre uma instalação de arte digital e dispositivos móveis com capacidades Bluetooth. Desta forma é possível, à instalação de arte digital usar conteúdos fornecidos pelo público, como por exemplo fotografias, vídeos, musica, texto, etc. A instalação tem também a possibilidade de enviar conteúdos para o público. Das várias hipóteses existentes para enviar conteúdos entre a instalação de arte digital e o publico, o Bluetooth tornou-se a escolha natural devido à inexistência de custo associado às transferências de conteúdos.

Ainda através do Bluetooth é possível fazer uma gestão de utilizadores. Esta aplicação fornece funcionalidades para saber quantos utilizadores estão presentes num determinado momento, qual o tipo de dispositivos que usam e as suas

características de interacção. Essas características definem se um utilizador é muito activo, interagindo frequentemente com a aplicação, se aceita ou rejeita conteúdos, ou ainda qual o horário em que esse utilizador interage mais com a instalação de arte digital. De forma análoga a aplicação DiABlu MailMan permite conhecer quais os ficheiros recebidos de um utilizador, os ficheiros que lhe foram enviados, e destes últimos quais os que foram aceites ou não. Toda esta informação pode ter interesse para a instalação de arte digital, e visto que é disponibilizada pela aplicação, cabe ao artista digital tirar o melhor partido dela.

De modo a comunicar com outras aplicações sobre as quais é criada a instalação de arte digital é usado o protocolo OSC. A escolha deste protocolo prende-se com o facto de já estar implementado nas principais aplicações com que o artista digital normalmente trabalha. Para facilitar o uso deste protocolo, e facilitar a integração com as ferramentas já referidas, foi definida um dialecto de mensagens, assim como os argumentos de cada uma. Deste modo é possível que, tanto a aplicação DiABlu MailMan como a instalação de arte digital comuniquem de uma forma inequívoca.

1.4 Organização e temas abordados

O presente relatório visa ilustrar os aspectos mais relevantes no desenvolvimento do projecto DiABlu MailMan. É neste capítulo introdutório que se apresenta o contexto onde o projecto está inserido, seguindo-se uma breve descrição do problema proposto. Neste capítulo é ainda feita uma análise superficial do projecto, assim como uma apresentação da estrutura do relatório. No seguinte capítulo é feita uma análise do estado da arte, e ainda um estudo das tecnologias que serão usadas. No terceiro capítulo está descrita toda a fase de análise e especificação dos requisitos, assim como o estudo da arquitectura do sistema. O quarto capítulo é dedicado à implementação. Nesse capítulo são expostos os detalhes mais importantes do projecto, que problemas levantaram, e como foram solucionados. O quinto capítulo é dedicado à análise do projecto após a sua conclusão, nomeadamente identificando o que foi e não foi concluído, assim como o resultado de testes realizados sobre a sua estabilidade e desempenho. No sexto capítulo é feito um resumo de tudo o que foi exposto ao longo deste relatório expondo-se também as principais conclusões do projecto, assim como os principais futuros desenvolvimentos

Capítulo 2

Análise do estado da arte

Este capítulo está dividido em duas partes. Na primeira parte é feito o estudo de soluções existentes, e na segunda parte faz-se uma análise das tecnologias utilizadas. No estudo das soluções existente é feita uma análise dos produtos que existem no mercado que poderiam solucionar o problema em mãos. Na análise tecnológica é feita uma descrição das tecnologias utilizadas, quais as suas características, e o porquê da sua escolha.

2.1 Estado da arte

A pesquisa de uma solução já existente para o problema quer comercial ou livre mostrou-se infrutífera, conforme seria de esperar. No entanto foram encontradas várias soluções com objectivos semelhantes, embora nenhuma esteja direccionada para a área da arte digital. Apresentam-se de seguida aquelas consideradas mais importantes e mais próximas do projecto realizado.

2.1.1 Bluetooth Factory da Shortcut

Figura 2.1 - Logótipo da Shortcut

O *Bluetooth Factory* [10] (BTF) apresenta-se como a solução da *Shortcut* [11] para marketing de proximidade. Pode definir-se como uma ferramenta de interacção

entre um dispositivo móvel e um servidor via Bluetooth, visando a troca de conteúdos e de informação.

O BTF é composto por dois módulos com objectivos diferentes. Um primeiro módulo permite realizar o envio de conteúdos publicitários de um *hotspot* para todos os dispositivos móveis que se encontrem dentro do alcance do mesmo. Um segundo módulo requer a existência de uma aplicação cliente nos dispositivos móveis que deverá permitir a comunicação com os servidores existentes nos *hotspots*. Este segundo módulo permite um marketing de proximidade mais efectivo, favorecendo a interacção entre os fornecedores de conteúdos e o público-alvo, o que representa uma vantagem para os fornecedores de conteúdos, já que fornece feedback imediato. Outra característica deste módulo é o facto de, do ponto de vista de marketing, não ser intrusivo para o cliente uma vez que a iniciativa para a comunicação está do lado deste.

2.1.2 BeamZone da Blue Cell Networks

Figura 2.2 - Logótipo da Blue Cell Networks

A *Blue Cell Networks* [12] (BCN) é uma empresa europeia dedicada a tecnologia de *hotspots* para telemóveis e PDA's. O *BeamZone* [13] da BCN é o primeiro sistema baseado em *hotspots* capaz de enviar conteúdos digitais gratuitos para dispositivos Bluetooth. Devido à rigidez do seu funcionamento baseado num sistema de permissões, o *BeamZone* é largamente aceite e o que o tornou muito popular.

O *BeamZone* é uma tecnologia pensada principalmente para campanhas de marketing utilizando a tecnologia Bluetooth. Constitui uma espécie de estação de entretenimento para dispositivos Bluetooth, orientado principalmente a telemóveis e PDAs. Isto é conseguido fornecendo diferentes tipos de conteúdos (informação, software, imagens, musica, vídeo, jogos, etc.) a dispositivos móveis através da tecnologia Bluetooth. O *BeamZone* identifica todos os dispositivos Bluetooth ao seu alcance, oferecendo automaticamente conteúdos compatíveis e apropriados. Os utilizadores interessados podem facilmente, após a respectiva autorização, receber os ficheiros nos seus dispositivos.

Os conteúdos disponíveis no *BeamZone* são sempre distribuídos de forma gratuita e são sempre entregues mediante a autorização do receptor. A tecnologia

usada pelo *BeamZone* permite que os conteúdos sejam rapidamente transferidos para os dispositivos. Todo o sistema é simples de usar e bastante intuitivo.

De modo a não ser uma tecnologia invasiva, ou seja, a não enviar conteúdos sem o conhecimento do público-alvo, foram considerados dois níveis de permissão. O primeiro nível depende totalmente do utilizador, pois só se este activar o Bluetooth do seu dispositivo é que o *BeamZone* pode iniciar o envio de conteúdos. O segundo nível de permissões prende-se com a aceitação, por parte do utilizador, do primeiro conteúdo que lhe será enviado. Caso o utilizador aceite este primeiro conteúdo, o *BeamZone* continuará a enviar conteúdos para o mesmo dispositivo; caso contrário, o mesmo não voltará a ser contactado.

2.1.3 Qwikker da Qwikker Corporation

O *Qwikker* [14] é uma plataforma local e social para distribuição de conteúdos para telemóveis que permite uma utilização fácil e gratuita. Com esta plataforma é possível efectuar o download e a partilha de conteúdos móveis tais como música, vídeos, jogos, aplicações Java, toques, informações, notícias, cupões e endereços de páginas na Internet. Esta plataforma é amplamente usada, e é responsável pela maior rede de distribuição de conteúdos utilizando a tecnologia Bluetooth, dispondo de mais de 800 pontos de acesso espalhados por todo o mundo, nos mais variados locais.

O *Qwikker* usa três formas diferentes de distribuição de conteúdos, aumentando significativamente a sua capacidade de colocar esses mesmos conteúdos nas mãos dos clientes alvo. A entrega pode ser feita de um servidor Bluetooth para um telemóvel, de um servidor Web para um telemóvel, ou ainda por partilha de conteúdos entre telemóveis. Enquanto os clientes se encontrarem no raio de alcance de um servidor Bluetooth, os conteúdos podem ser descarregados gratuitamente. Quando estiverem fora deste raio podem continuar a descarregar conteúdos utilizando a tecnologia WAP.

Figura 2.3 – Funcionamento do Qwikker

Para além de enviar conteúdos, o *Qwikker* constrói e entrega canais móveis e experiências interactivas com base em HTML possuindo capacidade transaccional e de resposta directa. Estes canais móveis podem incluir: um conjunto de páginas HTML desenhadas para ecrãs móveis, colecções de conteúdos móveis, grátis ou exclusivos, ligações *click-to-web* e *click-to-call*.

O *Qwikker* ajuda as marcas e as agências a desenhar campanhas para a entrega de conteúdos usando dispositivos móveis. A aplicação produzida em conjunto com as marcas possibilita a permanente exposição das mesmas nos telemóveis. Os conteúdos podem ser actualizados remotamente durante a campanha. No final da campanha são fornecidos às empresas relatórios de talhados que permitem ao fornecedor de conteúdos medir o retorno exacto do investimento efectuado.

2.1.4 Wilico da FuturLink

A *FuturLink* [15] é uma empresa dedicada à alta tecnologia orientada para desenvolver e inovar produtos e aplicações para interagir com telemóveis, usando tecnologias de curto alcance, tais como o Bluetooth e Wi-Fi.

O *Wilico* [16], um dos produtos da *FuturLink*, dispõe da tecnologia mais vanguardista nos seus *hotspots*, para oferecer a melhor experiência de interacção com telemóveis que se encontrem ao seu alcance, sendo uma ferramenta muito eficaz de marketing de proximidade com o objectivo de realizar campanhas de comunicação relacional ou emocional, comunicando produtos, serviços e promoções.

A tecnologia dos *hotspots Wilico* reconhece a maioria das marcas e modelos de telefones móveis com tecnologia Bluetooth, permitindo adaptar de forma automática os conteúdos às características de cada telemóvel, ao nível do sistema operativo, dimensão do ecrã e propriedades multimédia.

A cobertura dos *hotspots Wilico* é feita de forma gradual entre os 5 e os 100 metros e está testada em mais de 25 marcas e 550 modelos de telemóveis com capacidades Bluetooth.

Os *hotspots Wilico* incluem um software de criação e gestão de conteúdos, denominado *Suite Wilico*, que permite a criação de aplicações móveis interactivas (Java, Symbian e Windows Mobile) para uma ampla variedade de telefones móveis de forma automática, sem que para tal sejam necessários conhecimentos de programação. Estas aplicações podem ser estáticas ou dinâmicas e podem ser construídas através de esquemas pré-estabelecidos personalizáveis para cada empresa, segundo as suas necessidades.

2.1.5 Bluesender

O *BlueSender* [17] é uma aplicação ligada ao marketing de proximidade que permite o envio de ficheiros directamente para dispositivos móveis usando Bluetooth. A aplicação envia automaticamente ficheiros para todos os dispositivos com capacidades Bluetooth, como por exemplo telemóveis, PDAs e laptops, que se encontrem num alcance de 10 metros.

Tem como principal vantagem não necessitar de software específico nos dispositivos móveis. Além disso, a interacção é feita de uma forma não intrusiva (é necessário permissão do receptor para que os ficheiros sejam enviados), e cada ficheiro é enviado apenas uma vez para cada dispositivo, durante cada sessão. É ainda possível definir o intervalo de tempo no qual cada ficheiro deve ser enviado.

2.1.6 Bluetooth OBEX File Transfer

A aplicação *Bluetooth OBEX File Transfer* [18] permite gerir os ficheiros de um dispositivo que suporte o serviço *OBEX File Transfer Profile*, usando uma ligação Bluetooth. Permite fazer download, upload, explorar, apagar e criar ficheiros num telemóvel, PDA, Palm, laptop, etc. usando uma interface gráfica intuitiva, rápida e fácil de usar. Suporta ainda Drag & Drop e transferências de ficheiros em segundo plano, de modo a tornar a interacção mais fácil e mais semelhante ao que acontece, por exemplo, no Windows Explorer. Resumidamente, esta aplicação usa a tecnologia Bluetooth de modo a ser possível interagir com o telemóvel como se de um servidor de FTP se tratasse.

2.1.7 Conclusão da análise do estado da arte

O estudo do estado da arte revelou que não existe nenhuma aplicação semelhante à ferramenta que se pretende com DiABlu MailMan. À excepção do *Bluetooth OBEX File Transfer*, todas as aplicações encontradas são dedicadas ao marketing de proximidade, que apenas enviam conteúdos ao utilizador, mas não têm a capacidade de receber conteúdos enviados pelo utilizador. No que respeita à aplicação DiABlu MailMan grande parte do interesse da possibilidade de interacção em instalações de arte digital é o uso de conteúdos enviados pelo utilizador.

Outra grande lacuna nestas aplicações é a impossibilidade de serem controladas por uma aplicação externa. No caso da aplicação DiABlu MailMan, este controlo é conseguido recorrendo ao protocolo OSC. O OSC permite que aplicações que usem este protocolo controlem a aplicação DiABlu MailMan, ou seja, que através dela enviem e recebam ficheiros para dispositivos com capacidades Bluetooth, entre outras funcionalidades.

Algumas das aplicações estudadas usem aplicações que são executadas no telemóvel dos utilizadores. Estas aplicações podem levantar graves problemas de segurança, pois podem permitir acessos não autorizados aos dispositivos onde estão a ser executadas. Alem disso, é bastante difícil criar uma aplicação que seja compatível e executada de igual forma em todos os modelos de todas as marcas de dispositivos móveis com capacidades Bluetooth.

Por fim, a aplicação *Bluetooth OBEX File Transfer* está um pouco fora do que é pretendido deste projecto. Basicamente esta aplicação transforma um dispositivo com capacidades Bluetooth num servidor de ficheiros, o que não é o comportamento pretendido para a aplicação DiABlu MailMan. Realmente, o acesso a todos os ficheiros do dispositivo do utilizador poderia levantar problemas de segurança, alem de ser um tipo de acesso invasivo.

2.2 Tecnologias usadas

Neste capítulo é feita uma análise das tecnologias usadas no desenvolvimento da aplicação DiABlu MailMan. Além desta análise é ainda, neste capítulo, justificado o porquê da sua escolha. No caso de haver mais do que uma alternativa viável são expostos os factores nos quais as opções escolhidas são superiores às outras.

2.2.1 Java

Figura 2.4 - Duke, a mascote do Java

O Java [19] é uma linguagem de programação criada pela *Sun Microsystems* [20] lançada em 1996 como um componente principal da *Java platform* da mesma empresa. A sintaxe desta linguagem deriva do C e do C++, mas tem um modelo de objectos mais simples, assim como um número reduzido de funcionalidades de baixo nível. As aplicações Java são geralmente compiladas para um *bytecode* que é depois interpretado pela *Java Virtual Machine* (JVM) [21], independentemente da arquitectura do computador.

Os compiladores de Java originais e de referência, JVMs e bibliotecas de classes, foram desenvolvidos pela *Sun* em 1995. Em Maio de 2007, em

concordância com as especificações do *Java Comunity Proccess*, a Sun lançou a maior parte das tecnologias Java como software livre sobre uma *GNU General Public License*.

A criação desta linguagem de programação teve como motivação satisfazer os cinco seguintes objectivos:

- Deveria usar uma metodologia de programação orientada a objectos
- Deveria permitir que um mesmo programa fosse executado em múltiplos sistemas operativos
- Deveria conter suporte embutido para uso de redes de computadores
- Deveria ser desenhado para executar código de fontes remotas
- Deveria ser fácil de usar, aproveitando as melhores partes de outras linguagens de programação orientadas a objectos.

2.2.1.1 Portabilidade

A portabilidade significa que um programa criado usando a linguagem de programação Java deve correr de uma forma similar em qualquer hardware/sistema operativo que suporte a plataforma Java. Isto quer dizer que um programa só necessita de ser escrito uma vez, e depois de compilado poderia ser executado em qualquer lado.

Esta característica é conseguida pela maior parte dos compiladores de Java. O código é compilado para instruções máquina simplificadas específicas da plataforma Java (*Java bytecode*). O código é depois executado numa JVM, que é um programa escrito no código nativo do hardware que interpreta e executa *Java bytecode* genérico. Em algumas versões desta JVM, o *Java bytecode* pode ser compilado em código nativo, antes, ou durante a execução do programa, resultando em ganhos a nível de desempenho. Alem disso a uniformização das bibliotecas permite o acesso a funcionalidades das máquinas anfitrião (como por exemplo, gráficos, funcionalidades de rede e de threading) de uma maneira unificada.

As primeiras implementações da linguagem usavam uma JVM interpretada de modo a conseguir portabilidade. Estas implementações produziam programas mais lentos do que os que eram compilados para executáveis nativos, como era o caso do C e do C++. Consequentemente esta linguagem obteve a reputação de produzir programas com um baixo desempenho. As implementações das JVM recorrem a diversas técnicas para produzir programas que correm significativamente mais rápido.

Uma dessas técnicas conhecida como *just-in-time compilation* (JIT), traduz Java bytecode em linguagem nativa enquanto o programa está a correr, o que resulta num programa que é executado com melhor desempenho do que a solução de

código interpretado. Esta técnica, no entanto, é vítima de um overhead de compilação durante a execução.

As JVM mais sofisticadas usam outra técnica conhecida como *dynamic recompilation*, que consiste numa análise do comportamento do programa em execução, por parte da JVM, e selectivamente recompilar e optimizar as partes críticas desse mesmo programa. A *dynamic recompilation* consegue uma optimização superior à compilação estática, pois o compilador dinâmico consegue basear as optimizações no conhecimento do ambiente durante a execução, no conjunto de classes carregadas, e na identificação de *hotspots* (partes do programa, geralmente ciclos, que gastam a maior parte do tempo de processamento). O JIT e a *dynamic recompilation* conseguem tomar partido da velocidade do código nativo sem perder portabilidade.

Outra técnica, vulgarmente designada de *static compilation*, consiste em compilar directamente em código nativo, à semelhança dos compiladores tradicionais. Os compiladores de Java estáticos traduzem código Java em código objecto nativo, removendo o passo intermédio de compilação para *Java bytecode*. Esta solução produz desempenho superior, quando comparado com a versão interpretada, mas perde na portabilidade. O executável que resulta destes compiladores só pode ser usado numa única arquitectura.

2.2.1.2 Gestão automática de memória

Uma das ideias por trás do modelo de gestão automática de memória do Java é poupar os programadores do fardo de realizarem a gestão manual da memória. Em algumas linguagens de programação, o programador aloca memória para a criação de objectos guardados na *heap* e é responsável pela posterior libertação da memória. Se o programador se esquece de libertar a memória, ou de escrever código que o faça, ocorre um *memory leak*, o que geralmente resulta num consumo exagerado e arbitrário de memória. Além disso, se o programador tenta libertar mais do que uma vez a mesma região de memória, o funcionamento do programa pode tornar-se instável, pois esta acção tem consequências indefinidas. Finalmente, em ambientes onde não ocorre *garbage collection*, existe sempre um certo grau de overhead e de complexidade no código criado no sentido de detectar e libertar alocações.

No Java, este potencial problema é evitado pelo uso de *automatic garbage collection*. O programador determina quando é que os objectos são criados e o *Java runtime* é responsável por gerir o ciclo de vida dos objectos. O programa, ou outros objectos, pode referenciar um objecto guardado uma referência deste (o que a baixo nível não é mais do que guardar o seu endereço na *heap* de objectos). Quando não existem referências para um objecto, esse objecto torna-se inacessível, e é elegível

para ser libertado pelo *Java garbage collector*, ou seja pode ser libertado pelo *garbage collector* a qualquer momento. No entanto podem ainda ocorrer *memory leaks* se o código guardar uma referência para um objecto que já não é necessário. Por outras palavras, ainda podem acontecer *memory leaks*, mas apenas a níveis conceptuais mais elevados.

O uso da *garbage collection* numa linguagem pode afectar o seu paradigma de programação. Se, por exemplo, um analista/programador assume que o custo de alocação/libertação de memória é baixo, pode optar por criar objectos de uma forma mais liberal, em vez de os pré-inicializar, manter e reutilizar. Com um pequeno custo potencial ao nível do desempenho (criação de objectos grandes ou complexos dentro de ciclos), é possível facilitar o isolamento de *threads* (não existe a necessidade de sincronizar as *threads*, pois podem trabalhar em instâncias diferentes de um objecto).

É possível implementar uma funcionalidade semelhante em C++ (por exemplo implementar um modelo de gestão de memória para classes específicas para melhorar o desempenho e minimizar consideravelmente a fragmentação da memória), com o custo adicional de um overhead durante a execução comparável ao *garbage collector* do Java, e ainda de tempo de desenvolvimento e aumento da complexidade da aplicação, se não for usada uma biblioteca já existente. No Java, a *garbage collection* é invisível para o analista/programador, isto é não, existe a noção de quando é que a *garbage collection* será feita, pois não está necessariamente correlacionada com nenhuma acção executada pelo código escrito. Dependendo da aplicação pretendida pode ser benéfico ou prejudicial, pois apesar de libertar o programador da execução de tarefas de baixo nível, este perde a opção de escrever código de baixo nível.

O Java não suporta aritmética de apontadores, como é suportado por exemplo no C++. Isto deve-se ao facto de o *garbage collector* voltar a alocar objectos referenciados, invalidando assim os apontadores. Outra razão para a proibição prende-se com a impossibilidade de garantir segurança no caso de permitir manipulação arbitrária dos apontadores.

2.2.1.3 Porquê Java?

A linguagem de programação Java desde cedo se tornou uma das linguagens mais interessantes para o desenvolvimento deste projecto.

Em primeiro lugar, mas não de uma forma limitativa, está o facto de todas as aplicações que fazem parte do projecto DiABlu terem sido desenvolvidas em Java. Logo aqui se mostrou vantajoso a escolha desta linguagem, pois torna mais fácil a reutilização de código caso haja necessidade, além de manter a coerência entre as aplicações.

Outro dos principais factores, provavelmente o mais importante, é a portabilidade. Um dos requisitos da aplicação foi que a aplicação DiABlu MailMan pudesse ser executada de igual forma tanto no sistema operativo Microsoft Windows XP ou superior como no sistema operativo Mac OS X.

Foi também de considerar nesta escolha a familiaridade existente com a linguagem, assim como a excelente documentação existente além de que possui bibliotecas que implementam os protocolos de comunicação Bluetooth e OSC. Estas bibliotecas serão posteriormente apresentadas no decorrer deste capítulo.

2.2.2 Bluetooth

O Bluetooth é uma tecnologia rádio de curto alcance, de baixo custo e de baixa potência. Foi criada com a intenção de substituir as ligações por cabo entre telemóveis, PDAs e outros dispositivos portáteis. A *Ericsson Mobile Communications* foi a pioneira no desenvolvimento deste sistema em 1994, em busca de um substituto para os cabos que ligavam os telemóveis aos seus acessórios. Os primeiros dispositivos Bluetooth apareceram no mercado no ano de 1999. O nome Bluetooth vem de *Harald Blåtand*, um rei Viking da Dinamarquês que uniu e controlou a Noruega e a Dinamarca

O *Bluetooth Special Interest Group* (SIG) [22] é o responsável pelo continuar do desenvolvimento da tecnologia Bluetooth, sendo a A Sony Ericsson, Intel, IBM, Toshiba, Nokia, Microsoft, 3COM e a Motorola, são algumas das companhias envolvidas neste SIG. O Bluetooth SIG é uma das grandes vantagens da tecnologia Bluetooth, pois a participação de um conjunto de fabricantes de software e hardware importantes no desenvolvimento desta tecnologia garante que os dispositivos Bluetooth são disponibilizados para o utilizador final. A Microsoft fornece suporte Bluetooth através do seu sistema operativo *Microsoft Windows*, de maneira que esta tecnologia está disponível para uma grande parte do mercado de software; a Intel está a incluir a tecnologia Bluetooth nas suas Motherboards, principalmente nas que são usadas em computadores portáteis, e tanto a Nokia como a Sony Ericsson, incluíram esta tecnologia nos seus telemóveis mais recentes. Tudo isto contribui para uma grande disponibilidade da tecnologia Bluetooth para o utilizador final.

2.2.2.1 Arquitectura Bluetooth

O objectivo da especificação Bluetooth é permitir que dispositivos Bluetooth de diferentes fabricantes sejam compatíveis entre si, mesmo além da especificação do sistema de rádio. Como consequência, a especificação Bluetooth, não só faz o traçado de um sistema rádio, como define uma *stack* de protocolos de modo a

assegurar que os dispositivos Bluetooth conseguem descobrir outros dispositivos e explorar e usar os seus serviços disponíveis.

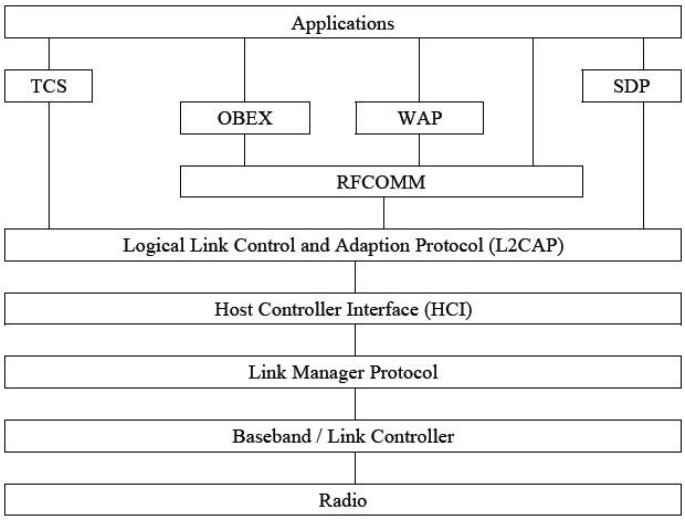

Figura 2.5 - A *stack* do protocolo Bluetooth

A *stack* Bluetooth é composta por várias camadas, como ilustra a figura 2.5. A camada HCI é geralmente a camada que separa o software do hardware, e é parcialmente implementada em software e hardware/firmware. Na sua generalidade, as camadas abaixo da HCI são implementadas em hardware, e as camadas superiores em software. É de notar que alguns dispositivos com funcionalidades limitadas, como por exemplo os auriculares, podem ter as suas funcionalidades completamente implementadas em hardware/firmware.

Tabela 2.1 - Descrição das camadas do protocolo Bluetooth

Camada	Descrição
Application	Os perfis Bluetooth guiam os analistas/programadores na forma como deve ser usada a pilha Bluetooth
Telephony Control System (TCS)	Fornece serviços de telefonia
Service Discovery Protocol (SDP)	É usada para a descoberta de serviços em dispositivos remotos
WAP e OBEX	Fornecem interfaces para partes de camadas mais altas de outros protocolos de comunicação
RFCOMM	Fornece uma interface serie similar à RS-232
L2CAP	Faz a multiplexagem de dados de camadas superiores e faz a conversão entre pacotes de diferentes tamanhos.
HCI	Trata da comunicação entre o anfitrião e o módulo Bluetooth
Link manager Protocol	Controla e configura ligações para outros dispositivos
Baseband e Link Controller	Controla ligações físicas, saltos de frequência e a assemblagem de pacotes
Radio	Trata da modulação e demodulação de dados para transmissão

	e recepção através do ar

Os analistas/programadores, ao criarem aplicações, não necessitam de saber todos os detalhes das camadas da *stack* Bluetooth. No entanto, é importante conhecer o funcionamento do rádio. O rádio Bluetooth é a camada mais baixa da comunicação por Bluetooth e funciona na banda *Industrial, Scientific and Medic* (ISM) na frequência dos 2.4 GHz. Existem várias tecnologias a usar esta banda, pelo que tecnologias Wi-Fi como é o caso do IEEE 802.11b/g e alguns aparelhos de cozinha, como por exemplo fornos micro-ondas, podem causar interferências nesta banda.

O rádio Bluetooth utiliza uma técnica de sinalização chamada *Frequency Hopping Spread Specturm* (FHSS). A banda rádio é dividida em 79 sub-canais, e o rádio Bluetooth usa um destes canais a cada momento. O rádio salta de canal em canal, ficando 625 micro segundos em cada um, o que resulta em 1600 saltos de frequência por segundo. Esta técnica é usada de forma a reduzir a interferência causada por dispositivos Bluetooth que se encontrem nas imediações, ou de outros dispositivos que utilizem a mesma banda.

A cada dispositivo Bluetooth é atribuído um endereço Bluetooth único. É um endereço de hardware de 48 bits equivalente aos endereços de hardware atribuídos às *Network Interface Cards* (NICs). O endereço Bluetooth é usado para identificação, sincronização dos saltos de frequência entre dispositivos e geração de chaves nos procedimentos de segurança do Bluetooth.

2.2.2.2 Bluetooth network

O tipo mais comum de rede Bluetooth chama-se *piconet*. É constituída por um dispositivo *master* e por um ou mais dispositivos *slave* activos, sendo no máximo sete. Ao dispositivo que inicia a comunicação é automaticamente atribuída a designação de *master*. O dispositivo *master* é aquele que controla a piconet. Os dispositivos *slave* só podem efectuar transferência de dados quando lhes é atribuído tempo de transferência pelo dispositivo *master*. Além disso, esses dispositivos não podem comunicar directamente entre si, pelo que todas as comunicações têm de passar obrigatoriamente pelo dispositivo *master*. Os dispositivos *slave* sincronizam a sua frequência de salto usando o sinal e endereço do dispositivo *master*.

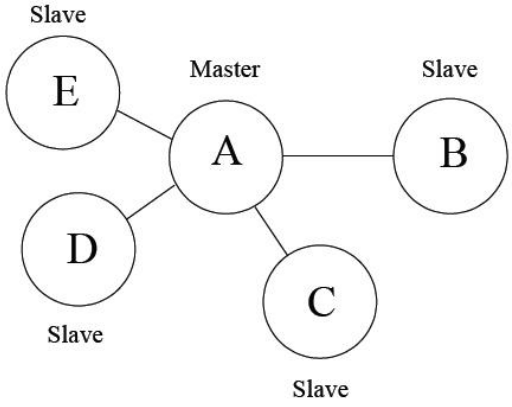

Figura 2.6 - Exemplo de uma *piconet* típica

Assim, as *piconet* têm a forma de uma rede em estrela, com o dispositivo *master* a ocupar o nó central, como é ilustrado na figura 2.6. Podem existir várias *piconets* ao alcance umas das outras mas, como os saltos de frequência não estão sincronizados, elas irão eventualmente colidir na mesma frequência.

Quando se faz a ligação de duas *piconets*, obtem-se uma *scatternet*. Na figura 2.7 apresenta-se um exemplo de uma destas redes, com um nó intermédio a estabelecer a ligação entre duas *piconets*. O nó intermédio tem de usar técnicas de time-sharing, o que significa que tem de seguir os saltos de frequência de cada *piconet* separadamente. Vai haver então uma redução do número de intervalos de tempo alocados entre o nó intermédio e o *master* de cada *piconet*, o que vai reduzir a taxa de transferência do nó intermédio para metade. É importante notar que nem a versão 1.1 nem a versão 1.2 da especificação do Bluetooth definem como os pacotes devem ser encaminhados entre *piconets*, razão pela qual estas comunicações não são fiáveis.

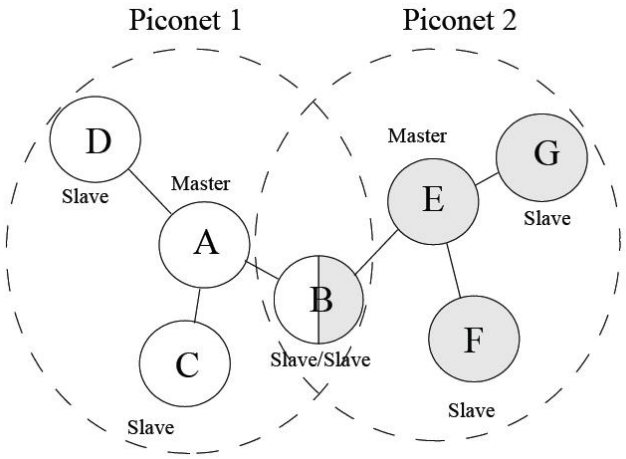

Figura 2.7 - Exemplo de uma *scatternet*

É permitido que dois dispositivos troquem de papéis dentro de uma rede Bluetooth, ou seja, um dispositivo *master* passa a *slave* e um dispositivo *slave* passa

a *master*. Considere-se o seguinte exemplo ilustrado pela figura 2.7 Um dispositivo A liga-se a um dispositivo B criando uma *piconet*. Como a ligação foi iniciada pelo dispositivo A, este dispositivo é o *master* da piconet, sendo o dispositivo B o *slave*.

Considere-se agora que um terceiro dispositivo B se quer juntar à *piconet*. Esse dispositivo vai estabelecer uma ligação com o dispositivo A. Como é o dispositivo C que inicia a ligação, vai ser ele o *master* da ligação entre os dispositivos A e C. O dispositivo A passa então a ser, simultaneamente, *master* da ligação entre os dispositivos A e B e *slave* da ligação entre os dispositivo A e C. O que se obtem desta ligação é uma *scatternet*.

Se se efectuar a troca de papeis entre os dispositivos A e C, transforma-se a *scatternet* numa *piconet*, sendo o dispositivo A o *master* e os dispositivos B e C os *slaves* da piconet. Compreende-se então a importância da troca de papéis, pois é necessário recorrer a esta técnica cada vez que um dispositivo se pretende ligar a uma *piconet* existente. Se tal não ocorrer contrário obtêm-se uma *scatternet*.

2.2.2.3 Ligações Bluetooth

Na especificação Bluetooth estão definidos dois tipos de ligações, *Synchronous Connection Oriented* (SCO) e *Asynchronous ConnectionLess* (ACL). As ligações SCO e ACL fazem parte da especificação da camada *Baseband*

As ligações SCO são usadas em transmissões de áudio. Quando uma ligação é estabelecida, são reservados intervalos de tempo para transmissão de dados, obtendo uma *Quality of Service* (QoS) garantida. Os pacotes perdidos ou com erros não são retransmitidos, o que faz todo o sentido para transmissão de voz. Todas as ligações SCO funcionam a 64 kbps. Um dispositivo *master* pode ter até três ligações SCO em simultâneo, para o mesmo ou diferentes dispositivos *slave*. Um dispositivo *slave* pode ter três ligações SCO simultâneas com um dispositivo *master*.

As ligações ACL têm a transferência de dados como principal uso. Uma ligação ACL fornece uma transmissão de dados sem erros, o que significa que pacotes com erro, ou pacotes perdidos, são retransmitidos, e como consequência, não há garantia de QoS. A taxa máxima de transferência para uma ligação ACL ronda os 650 kbps. Um dispositivo *master* pode ter quantas ligações ACL forem necessárias, mas apenas uma ligação ACL pode existir entre dois dispositivos. Os dados do utilizador são geralmente transmitidos de, e para a camada *Logical Link Control and Adaptation Protocol* (L2CAP) da *stack* Bluetooth. No desenvolvimento de aplicações é normal referir-se ligações L2CAP e RFCOMM como ligações Bluetooth, embora a L2CAP e a RFCOM sejam camadas diferentes da *stack* Bluetooth, que necessitam de uma ligação física ACL para a transmissão de dados.

A camada L2CAP é responsável pela multiplexagem entre protocolos de camadas superiores sobre uma única ligação física ACL, permitindo que várias ligações lógicas possam ser estabelecidas entre dois dispositivos Bluetooth. A camada L2CAP fornece ainda funcionalidades de segmentação e posterior assemblagem de pacotes de camadas superiores. Os diferentes protocolos têm pacotes de diferentes tamanhos, e alguns deles necessitam de ser segmentados de modo a ser possível enviá-los através de uma ligação ACL, devido a restrições do tamanho dos pacotes. Um pacote ACL tem uma carga útil máxima de 339 bytes, enquanto um pacote L2CAP pode chegar aos 65,535 bytes de carga útil.

A camada RFCOMM emula uma porta série RS-232, assim como *streams* de dados. Esta camada depende do protocolo L2CAP para realizar a multiplexagem de múltiplos *streams* de dados concorrentes e para tratar das ligações com múltiplos dispositivos. A maior parte dos perfis Bluetooth usam o protocolo RFCOM por ser uma alternativa mais fácil do que interagir directamente com a camada L2CAP.

2.2.2.4 Descoberta de dispositivos e serviços

Devido à natureza *ad-hoc* das redes Bluetooth, dispositivos Bluetooth vão estar frequentemente a entrar e sair do seu alcance. Portanto torna-se necessário que os dispositivos Bluetooth tenham a capacidade de descobrir outros dispositivos Bluetooth que se encontrem ao seu alcance. Quando um dispositivo é descoberto, pode ser então iniciada uma descoberta de serviços, de modo a determinar quais os serviços fornecidos pelos dispositivo.

A especificação refere-se à descoberta de dispositivos como inquérito (*inquery*). Durante o processo de inquérito, o dispositivo que está a fazer a pesquisa recebe o endereço Bluetooth e a frequência de relógio dos dispositivos passíveis de serem descobertos, que se encontram nas imediações. Após a conclusão deste inquérito, o dispositivo que o fez é capaz de identificar outros dispositivos pelo seu endereço Bluetooth, e consegue sincronizar os saltos de frequência usado a frequência de relógio e o endereço Bluetooth dos dispositivos remotos.

Os dispositivos, para poderem ser descobertos, têm de entrar no modo *inquery scan.* Neste modo os saltos de frequência são mais lentos de que o normal, o que significa que o dispositivo vai passar mais tempo em cada canal, aumentando a probabilidade de encontrar um dispositivo que está a fazer um inquérito. Os dispositivos para estarem visíveis têm de usar um *Inquery Access Code* (IAC). Existem dois IACs diferentes, o *General Inquery Access Code* (GIAC), que é usado quando o dispositivo vai visível por um tempo indefinido, e o *Limited Inquery Access Code* (LIAC), quando o dispositivo vai estar visível por um período de tempo limitado.

Cada dispositivo Bluetooth pode oferecer diferentes conjuntos de serviços, o que torna necessário que um dispositivo faça uma pesquisa de serviços no dispositivo remoto, de modo a obter informação sobre os serviços disponíveis. As pesquisas de serviços podem ser generalistas, procurando todos os serviços disponíveis num determinado dispositivo, ou podem ser específicas, procurando apenas um serviço. O processo de descoberta de dispositivos usa o *Service Discovery Protocol* (SDP). Um cliente SDP tem de efectuar pedidos SDP a um servidor SDP para receber informação do registo de serviços do servidor.

2.2.2.5 Serviços Bluetooth

Os dispositivos Bluetooth contêm informação acerca dos serviços que fornecem numa *Service Discovery DataBase* (SDDB) conforme ilustrado na figura 2.8. O SDDB contem registos de cada serviço disponível. Estes registos são compostos por atributos que descrevem o serviço. Cada serviço disponível tem uma, e só uma, entrada na base de dados.

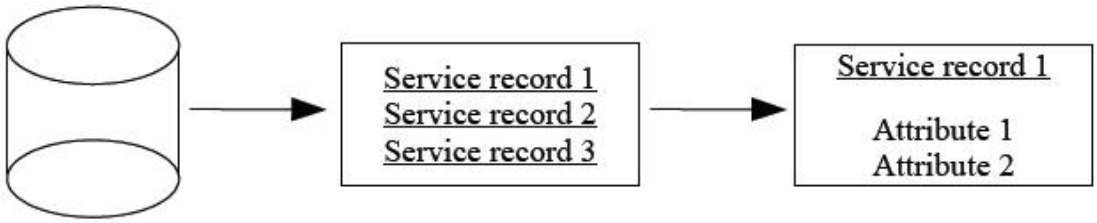

Figura 2.8 – *Service Discovery DataBase* (SDDB)

Os dispositivos remotos podem obter registos de serviços durante a pesquisa de serviços de modo a obter toda a informação necessária para usar esses mesmos serviços. Pode ser visto na figura 2.8 que um registo de um serviço contém vários atributos. Cada atributo é composto por um *attribute ID*, que é um identificador hexadecimal. Na tabela 2.2 são explanados os nomes dos atributos mais comuns, os seus IDs e o tipo de dados neles contidos. Apenas dois dos atributos são considerados necessários num registo de um serviço, e são eles o *ServiceRecordHandle* (com o ID 0x0000) e o *ServiceClassIDList* (com o ID 0x0001). Geralmente existem vários atributos adicionais nos registos dos serviços com o objectivo de descrever melhor os mesmos.

Tabela 2.2 - Atributos do registo de serviços

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer
ServiceClassIDList	0x0001	Sequência de *Data Elements* (de UUIDs)
ServiceRecordState	0x0002	32-bit unsigned integer
ServiceID	0x0003	UUID
ProtocolDescriptorList	0x0004	Sequência de *Data Elements* (de UUIDs e parâmetros específicos do protocolo) ou *Data Element* alternativo
BrowseGroupList	0x0002	Sequência de *Data Elements* (de UUIDs)
LanguageBaseAttributeIDList	0x0006	Sequência de *Data Elements*
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer
ServiceAvailability	0x0008	8-bit unsigned integer
BluetoohProfileDescriptorList	0x0009	Sequência de *Data Elements* (de UUIDs)
DocumentationURL	0x000A	URL
ClientExecutableURL	0x000B	URL
IconURL	0x000C	URL

Os diferentes atributos contêm valores de vários tipos e tamanhos. De modo a simplificar o uso destes atributos são usados *Data Elements* para guardar estes valores. Um *Data Elemen*t está dividido num campo descritor dos dados e num campo de dados. O campo descritor contém informação sobre o tipo e tamanho dos dados contidos no campo de dados do Data Element e o campo de dados contém os dados em si. O dispositivo remoto saberá assim qual o tipo de dados, e qual o tamanho dos dados que está a receber.

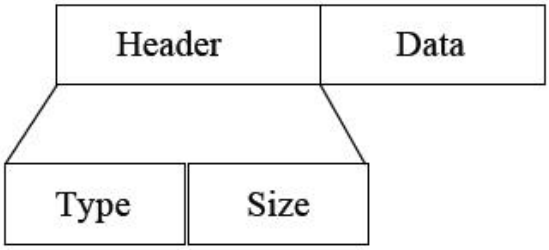

Figura 2.9 - Estrutura de um Data Element

O *Univesally Unique IDentifier* (UUID), é o tipo de dados usado na identificação de serviços, protocolos, perfis, etc. O UUID é um identificador de 128 bits, que tem a garantia de ser único. A tecnologia Bluetooth usa diferentes variantes de UUIDs (UUIDs curtos e UUIDs longos) de modo a reduzir o transtorno

de transferir valores de UUIDs de 128 bits. Uma gama de UUIDs curtos foi pré-alocada para os serviços, protocolos e perfis mais usados. Esta lista pode ser consultada no documento *Bluetooth Assigned Numbers*.

2.2.2.6 Perfis Bluetooth

Os perfis Bluetooth fornecem um conjunto bem definido de procedimentos de camadas superiores, assim como uma forma uniformizada de usar as camadas inferiores do Bluetooth. A tabela 2.3 mostra estes perfis, assim como uma breve descrição de cada um.

Tabela 2.3 - Perfis Bluetooth

Perfil	Descrição
Generic Access Profile (GAP)	É a base de todos os perfis no sistema bluetooth. O GAP define as funcionalidades básicas, como a inicialização de ligações L2CAP e o controlo dos modos de descoberta e segurança
Serial Port Profile (SPP)	Fornece emulação da porta série (RS-232) baseada na camada RFCOMM da stack Bluetooth
Dial Up Networking Profile (DUNP)	Define as funcionalidades necessárias para o uso de um dispositivo Bluetooth como uma gateway de ligação a uma rede
FAX Profile	Define as funcionalidades necessárias para usar um dispositivo Bluetooth como uma gateway de FAX
Headset Profile	Define as funcionalidades necessárias para transferência de áudio, por exemplo com um auricular Bluetooth
LAN Access Point Profile	Define as funcionalidades necessárias para usar um dispositivo bluetooth como um ponto de acesso à rede
Generic Object Exchane Profile (GOEP)	Fornece suporte para o protocolo OBject EXchange (OBEX) sobre uma ligação Bluetooth
Object Push Profile	Define funcionalidades para a troca de objectos vCard e vCalender, baseado em GOEP
File Transfer Profile	Define funcionalidades para navegar no sistema de ficheiros assim como copiar, apagar e criar ficheiros ou pastas num dispositivo Bluetooth, baseado em GEOP
Syncronization Profile	Define funcionalidades para sincronizar *Object Stores* contendo objectos IrMC (vCard, vCalendar, vMessaging and vNotes) entre dispositivos Bluetooth
Intercom Profile	Permite aos dispositivos Bluetooth estabelecer uma ligação de comunicação directa similar às comunicações entre intercomunicadores.
The Cordless Telephony Profile	Permite aos dispositivos Bluetooth funcionar como um telefone sem fios, por exemplo através de uma gateway ISDN.

O uso destes perfis garante a interoperabilidade entre diferentes dispositivos de diferentes *Original Equipment Manufacturers* (OEMs). É assim possível aos consumidores comprarem um telemóvel de uma marca e um auricular de outra com a certeza de que estes são completamente compatíveis, assumindo que ambos os dispositivos implementam o *Headset Profile*.

2.2.2.7 Limitações

A tecnologia Bluetooth possui algumas limitações, nomeadamente as relacionadas com o alcance e a taxa de transferência, que se apresentam nas tabelas seguintes

Tabela 2.4 - Alcance e potência das diferentes classes dos rádios Bluetooth

Classe	Potência máxima	Alcance
1	100 mW (20dBm)	~100 metros
2	2.5 mW (4dBm)	~10 metros
3	1 mW (0dBm)	~1 metro

O alcance de um dispositivo Bluetooth depende da classe do seu rádio. Por isso, o alcance escolhido para a maioria dos dispositivos móveis é apenas 10 metros, já que esta distância na prática é considerada suficiente para satisfazer o propósito do Bluetooth de substituir os cabos para ligações entre dispositivos. Esta distância tem também a vantagem de necessitar apenas de 2,5 mW para o seu funcionamento

Existem no entanto versões com maior alcance usadas principalmente em ambientes industriais que conseguem ultrapassar os 100 metros, mas para tal necessitam de muito mais potência. Isto torna pouco viável o seu uso em dispositivos móveis.

Em geral, e dado que a comunicação é estabelecida nos dois sentidos, o alcance é sempre igual ao dispositivo de menor potência

Outra limitação é a baixa taxa de transferência do Bluetooth. Os dispositivos mais recentes são capazes de transmitir e receber dados a velocidades próximas dos 3.0 Mbps em condições ideais, mas os dispositivos que apenas respeitam a primeira especificação da tecnologia conseguem somente velocidades de 720 kbps. Estas velocidades são consideradas baixas quando comparadas com os 11 Mbps ou mesmo os 54 Mbps oferecidos pela tecnologia Wi-Fi, que já está disponível em alguns dos dispositivos móveis topo de gama.

Na maioria dos casos, esta taxa de transferência é suficiente para a comunicação, pois as transferências feitas por Bluetooth tendem a ser de pequena dimensão. No caso de ser necessário transferir algo de maior dimensão, torna-se

necessário que o utilizador se mantenha dentro do alcance durante alguns minutos, até que a ligação termine.

As taxas de transferência mencionadas são partilhadas por todas as comunicações que estejam a decorrer com um determinado dispositivo, o que significa que a taxa de transferência efectiva seja quase sempre inferior à máxima. Esta taxa de transferência é também afectada pela distância e por obstáculos que possam existir entre os dispositivos em comunicação.

Outra das limitações desta tecnologia é o facto apenas permitir um número reduzido de ligações por dispositivo. Um dispositivo pode, no máximo, estar ligado a 7 outros dispositivos. Esta limitação tem maior importância quando é necessário enviar um ou mais ficheiros para todos os dispositivos Bluetooth dentro do raio de alcance. Por outro lado esta limitação é necessária, pois como já foi dito, a taxa de transferência é partilhada por todas as ligações. Aumentar demasiado o número de ligações activas, reduz a velocidade de cada uma dessas ligações, o que torna cada uma das transferências mais demorada. Além disso, mais ligações significam mais transferência de dados de *overhead*, o que baixa ainda mais a taxa de transferência efectiva.

2.2.2.8 Segurança

A tecnologia sem fios Bluetooth tem, desde o início, colocado um grande ênfase em questões de segurança, de modo a que os utilizadores desta especificação se sintam seguros quando a usam.

A segurança disponível em Bluetooth começa pelo estado de visibilidade em que se encontra o dispositivo. A especificação determina que um dispositivo Bluetooth, estando ligado, pode ou não estar visível para os outros. Se não estiver visível, o Bluetooth pode ainda assim ser usado para ligações a acessórios tais como sistemas mãos livres que tenham sido previamente conectados ao dispositivo.

2.2.2.9 Porquê Bluetooth?

O uso de Bluetooth neste projecto era um requisito obrigatório, por fazer parte do projecto DiABlu. No entanto, o uso desta tecnologia tem inúmeras vantagens. É uma tecnologia implementada na maior parte dos telemóveis mais recentes, PDAs e portáteis, que são o publico alvo da aplicação DiABlu MailMan. Além disso esta tecnologia, quando usada para a transferência de ficheiros, não tem encargos para nenhuma das partes envolvida na transferência, o que não acontece no envio de SMS e MMS, ou muitas vezes Wi-Fi..

2.2.3 OBEX

OBEX [23] é um acrónimo que significa *OBject EXchage*. É um protocolo de comunicação largamente usado na área das tecnologias da comunicação. Como é um protocolo mantido pela *Infrared Data Assocciation*, por vezes é conhecido como *IrOBEX* ou *Infrared OBEX*.

Este protocolo foi desenhado com o objectivo de trocar objectos binários entre dispositivos. Isto significa que o OBEX está desenhado para funcionar de forma semelhante ao HTTP, pois permite que um utilizador se ligue a um servidor de uma maneira análoga. Uma vez ligado a um servidor, o utilizador pode pedir ou fornecer objectos.

O OBEX está normalmente associado e optimizado para ligações sem fios de uma natureza *ad-hoc*. No entanto, no passado, o OBEX estava apenas reservado para ligações por infravermelhos. Hoje em dia o OBEX pode ser usado na maior parte das tecnologias de comunicação, como por exemplo TCP/IP e Bluetooth. Este protocolo está presente na grande maioria dos dispositivos de comunicação (os PDAs, como por exemplo os Palm Pilot, foram os primeiros a suportar o protocolo OBEX). Este protocolo pode ser encontrado também na maior parte dos telemóveis mais recentes, principalmente nos telemóveis produzidos pela Sony-Ericsson, Nokia, e Siemens e é suportado pelos sistemas operativos Microsoft Windows XP e superior e Mac OS X.

O protocolo OBEX é por vezes apelidado de *session protocol* ou *binary http protocol*. A transmissão binária permite a troca de informação relativa a um dado pedido ou objecto. Devido à capacidade de suportar sessões, o OBEX pode ramificar uma única ligação de transporte em várias operações correlacionadas. Por outras palavras, se uma transacção já foi fechada, o OBEX tem as funcionalidades necessárias para a voltar a abrir e continuar a transacção, de modo a que toda a informação permaneça intacta.

2.2.3.1 Porquê OBEX?

O OBEX é um protocolo de comunicação orientado à transferência de ficheiros em modo binário, que, entre outras formas, funciona sobre Bluetooth. Visto que é suportado pelos sistemas operativos alvo da aplicação DiABlu MailMan, e é vastamente usado nos dispositivos móveis com capacidades Bluetooth, tornou-se na escolha ideal para este projecto.

2.2.4 Open Sound Control

O Open Sound Control é um protocolo de comunicação entre computadores, sintetizadores de som e outros dispositivos multimédia. É um protocolo que está optimizado para a mais moderna tecnologia de redes e permite incluir as vantagens

da tecnologia de redes de computadores no mundo da arte digital. Entre as grandes vantagens deste protocolo estão a flexibilidade, interoperabilidade, eficácia, boa organização e documentação.

O OSC é um protocolo aberto, baseado em mensagens, e independente do tipo de transporte.

2.2.4.1 Tipos de Dados

Todos os dados OSC são compostos pelos seguintes tipos de dados: int32, float32, OSC-timetag, OSC-string e OSC-blob.

Destes cinco tipos de dados diferentes vale a pena explicar o significado e constituição dos últimos três:

- **OSC-timetag** - É a representação de um instante de tempo, com uma precisão de 200 picosegundos. É representado em 64 bits usando uma notação de vírgula fixa.
- **OSC-string** - É uma sequência de caracteres ASCII não nulos seguido de um null e de um conjunto de 0 a 3 caracteres nulos de modo a que o total de bits seja múltiplo de 32. Neste documento uma OSC-string será escrita sem os caracteres nulos e delimitada por aspas como por exemplo: “string”.
- **OSC-blob** – É constituído por um campo com um int32, que indica o numero de bytes do *blob*, seguido por esse mesmo número de bytes. No final são acrescentados 0 a três bytes nulos, de modo a que o número de bits do *blob* seja múltiplo de 32.

O tamanho de qualquer tipo de dados atómico em OSC é sempre multipo de 32. Assim é garantido que se o início do bloco de dados OSC estiver alinhado em 32 bit, qualquer tipo de dados está alinhado em 32 bits.

2.2.4.2 Pacotes OSC

As unidades de transmissão do protocolo OSC são os pacotes OSC. Qualquer aplicação que envie pacotes OSC é um cliente OSC e qualquer aplicação que receba pacotes OSC é um servidor OSC.

Cada pacote OSC é composto pelos seus conteúdos num bloco contínuo de dados binários, e o seu tamanho. Ou seja o número de bytes de 8 bits do conteúdo. O tamanho de um pacote OSC é sempre múltiplo de quatro.

A rede que transporta os pacotes OSC é responsável por entregar, tanto os conteúdos como o tamanho do pacote à aplicação OSC. Um pacote OSC pode naturalmente ser representado por um datagrama num protocolo de redes como é o caso do protocolo UDP. No caso do TCP, os dados devem começar com o int32 que

representa o tamanho do primeiro pacote, seguido do primeiro pacote, seguido pelo tamanho do segundo pacote, etc.

Os conteúdos dos pacotes OSC podem ser uma mensagem OSC ou um *bundle* OSC. O primeiro byte dos conteúdos do pacote deve distinguir entre estas duas alternativas, de uma forma não ambígua.

2.2.4.3 Mensagens OSC

Uma mensagem OSC é composta por um *OSC Address Pattern*, um *OSC Type Tag String*, e um ou mais argumentos OSC.

Em algumas implementações mais antigas do OSC é possível omitir a *OSC Type String*. Até essas implementações estarem actualizadas, todas as implementações devem ser robustas de modo a conseguirem tratar a mensagem *OSC* no caso de não existir uma *OSC Type Tag String*.

2.2.4.4 OSC Type Tag String

Uma *OSC Type Tag String* é uma OSC-string começada com o carácter ‘,’ seguida de sequência de caracteres que corresponde exactamente à sequência de argumentos OSC numa dada mensagem. Cada carácter posterior à vírgula é um *OSC Type Tag* e representa o tipo de dados do argumento OSC correspondente. O facto das *OSC Type Tag Strings* começarem com vírgula facilita a tarefa de verificar se uma mensagem OSC contém ou não uma *OSC Type String.*

Tabela 2.5 - O significado de cada *OSC Type Tag*

OSC Type Tag	Type of corresponding argument
i	int32
f	float32
s	OSC-string
b	OSC-blob

Algumas aplicações OSC comunicam entre instâncias delas mesmas com tipos de argumentos adicionais que não os definidos na tabela 2.4. As aplicações OSC não necessitam de reconhecer esses tipos. Qualquer mensagem que contenha uma *OSC Type Tag String* com *OSC Type Tags* desconhecidas deve ser ignorada.

2.2.4.5 Bundles OSC

Um *bundle* OSC é composto pela OSC-string “#bundle” seguido por uma OSC--timetag, e por zero ou mais elementos do *bundle* OSC. Um elemento do *bundle* OSC consiste num int32 que indica qual o número de bytes do conteúdo (que tem

de ser sempre múltiplo de 4), seguido desse conteúdo. Os conteúdos de um elemento de um *bundle* OSC podem ser tanto uma mensagem OSC como outro *bundle* OSC.

A tabela seguinte mostra a composição de um bundle OSC com dois ou mais elementos.

Tabela 2.6 - Composição de um *bundle* OSC

Dados	Tamanho	Função
OSC-string "#bundle"	8 bytes	Identifica um bundle OSC
OSC-timetag	8 bytes	OSC-timetag aplicada a todo o bundle OSC
Tamanho do primeiro elemento do bundle OSC	int32 = 4 bytes	Primeiro elemento do bundle OSC
Conteúdo do primeiro elemento do bundle OSC	Tantos bytes quantos definidos pelo "Tamanho do primeiro elemento do bundle OSC"	
Tamanho do segundo elemento do bundle OSC	int32 = 4 bytes	Segundo elementos do bundle OSC
Conteúdo do segundo elemento do bundle OSC	Tantos bytes quantos definidos pelo "Tamanho do segundo elemento do bundle OSC"	
etc.		Elementos adicionais do bundle OSC

2.2.4.6 OSC Address Spaces e endereços OSC

Todos os servidores OSC tem um conjunto específico métodos OSC. Os métodos OSC são os potenciais destinatários das mensagens OSC recebidas pelo servidor OSC, e correspondem a cada um dos pontos de controlo disponibilizados pela aplicação. Invocar um método OSC é análogo a uma chamada a um procedimento, ou seja, trata-se de fornecer argumentos a um método e fazer com que os eventos causados por esse método sejam executados.

Os métodos de um servidor OSC estão organizados numa estrutura de árvore chamada *OSC Address Space*. As folhas desta árvore são os métodos OSC, e os nós são chamados *OSC Containers*. Um *OSC Address Space* de um servidor OSC pode ser dinâmico, o que significa que a sua forma e conteúdos podem ser alterados ao longo do tempo.

Cada método OSC e cada *OSC Container* que não a raiz da árvore têm um nome simbólico. O endereço OSC de um método OSC é um nome simbólico que descreve

o caminho completo do método OSC no espaço de endereços OSC, começando na raiz da árvore.

O endereço de um método OSC começa com o carácter '/' seguido do nome de todos os *OSC Containers*, ordenados pelo seu nível na árvore, separados por caracteres '/' e, finalmente, o nome do método OSC. A sintaxe dos endereços OSC foi escolhida de modo a ser semelhante à sintaxe dos URLs.

2.2.4.7 Despacho de mensagem OSC e correspondência de padrões

Quando um servidor OSC recebe uma mensagem OSC, tem de invocar os métodos OSC apropriados existentes no seu *OSC Address Space* baseado no *OSC Address Pattern* da mensagem OSC. Este processo é o despacho da mensagem OSC para os métodos OSC que correspondem com o *OSC Address Pattern* da mensagem. Todos os métodos OSC são invocados com os mesmos tipos de argumentos, que são os argumentos OSC da mensagem OSC.

As partes de um endereço OSC ou de um *OSC Address Pattern* são as substrings entre os pares adjacentes de caracteres '/' e a substring a seguir ao ultimo carácter '/'.

O despacho da mensagem OSC recebida tem de ser feito para todos os métodos OSC no *OSC Address Space* actual, cujos endereços OSC correspondem ao *OSC Adress Pattern* da mensagem OSC. Um *OSC Adress Pattern* corresponde a um endereço OSC se:

- O endereço OSC e o *OSC Adresse Pattern* são compostos pelo mesmo número de partes
- Cada parte do *OSC Address Pattern* corresponde à mesma parte do endereço OSC.

2.2.4.8 Semântica temporal e OSC Time Tags

Um servidor OSC tem de ter acesso a uma correcta representação do tempo absoluto sempre que necessário. O OSC não fornece quaisquer mecanismos para sincronização de relógios.

Quando um pacote OSC recebido apenas contém uma única mensagem OSC, o servidor OSC deve invocar os métodos correspondentes imediatamente, ou pelo menos logo que possível após a recepção do pacote. Se um pacote OSC recebido contém um bundle de mensagens OSC, a *OSC Time Tag* do bundle OSC determina quando é que os métodos correspondentes devem ser invocados. Se o tempo representado pela *OSC Time Tag* é anterior ou igual ao tempo actual, o servidor OSC deve invocar os métodos o mais prontamente possível (excepto nos casos em que o servidor foi configurado para ignorar mensagens que cheguem demasiado

tarde). No caso contrário o servidor OSC deve esperar pelo instante representado pela *OSC Time Tag* para invocar os métodos OSC adequados. Até esse instante o servidor tem de guardar o bundle OSC

As *OSC Time Tags* são representadas por um número de vírgula fixa de 64 bits. Os primeiros 32 bits especificam o número de segundos desde 1 de Janeiro de 1900, e os últimos 32 bits representam fracções de segundo com uma precisão de cerca de 200 picosegundos. Esta é a representação nos *timestamps* da Internet NTP. Existe um caso especial nesta representação, que ocorre quando os 63 bits mais significativos são '0', e o bit menos significativo é '1'. Este caso significa que os métodos OSC devem ser invocado imediatamente.

As mensagens OSC dentro do mesmo *bundle* OSC são consideradas atómicas, ou seja, os seus métodos OSC correspondentes devem ser invocados imediatamente e em sucessão, e nenhum outro processo deve ocorrer entre a invocação de métodos OSC sucessivos.

Quando um *OSC Address Pattern* é despachado para múltiplos métodos OSC, não existe uma ordem especificada pela qual os métodos devem ser invocados. No entanto quando um *bundle* OSC contém múltiplas mensagens OSC, os conjuntos de métodos OSC correspondentes às mensagens OSC devem ser invocados pela mesma ordem que as mensagens OSC aparecem no pacote OSC.

Quando os *bundles* OSC contêm outros *bundles* OSC, a *OSC Time Tag* do *bundle* contido deve ser superior ou igual à *OSC Time Tag* do *bundle* que o contém. A atomicidade requerida pelas mensagens OSC dentro do mesmo OSC *bundle* não se aplica a *bundles* OSC dentro de um *bundle* OSC.

2.3 Bibliotecas Usadas

2.3.1 Bluetooth

A *Java Comunity Process* definiu um standard, o JSR-82 [24] de modo a permitir a criação de aplicações em Java com funcionalidades Bluetooth. É um standard livre e não proprietário. A API do JSR-82 esconde toda a complexidade da stack do protocolo Bluetooth, expondo um conjunto simples de APIs de Java.

De modo usar o JSR-82 é necessário usar uma implementação. No decorrer deste projecto foram testadas duas implementações, a BlueCove a a AvetanaBluetooth.

2.3.1.1 BlueCove

O Bluecove [25]é uma implementação do JSR-82 para J2SE. Actualmente pode ser usada nas seguintes stacks Bluetooth: Mac OS X, WIDCOMM, BlueSoleil e a Microsoft Bluetooth stack que se encontra no Windows XP SP2 e superior. Na versão 2.0.3 desta implementação foi adicionado um módulo GPL que acrescenta suporte para Linux BlueZ. Foi originalmente desenvolvida pela *Intel Research* e é actualmente mantida por voluntários. O Bluecove é desenvolvido sob uma licença GNU LGPL.

2.3.1.2 AvetanaBluetooth

A implementação JSR-82 avetanaBluetooth [26] fornece funcionalidades a software Java de uma forma padronizada. Usando a avetanaBluetooth é possível criar serviços Bluetooth, procurar por dispositivos remotos ou estabelecer uma ligação a serviços desses mesmos dispositivos. Esta implementação pode ser usada em Windows 98SE, Windows XP SP2, Mac OS X, Linux e WM 2003SE.

Esta implementação é comercial, e uma licença válida para 3 dispositivos custa 25€

2.3.1.3 Conclusões

No início do projecto usou-se ambas as implementações, pois a troca entre as implementações é rápida e não causa perdas de tempo. No entanto alguns problemas surgiram à medida que o projecto avançou. A certa altura, aquando da criação de um serviço no dispositivo local, esse serviço não era detectado por alguns dos telemóveis usados. Este problema ocorre em ambas as implementações.

No entanto um contacto com o responsável do projecto BlueCove resultou na correcção de um erro existente na versão disponível. Com a nova versão do BlueCove, o erro existente na aplicação DiABlu MailMan desapareceu.

Nenhuma resposta foi obtida por parte da equipa que desenvolveu o avetanaBluetooth. Uma pesquisa mais aprofundada na documentação da API revelou que uma das funcionalidades necessárias à correcta criação de serviços no dispositivo local não estava ainda implementada.

O fraco suporte técnico aliado a problemas técnicos resultantes do uso da implementação JSR-82, avetanaBluetooth, levaram ao seu abandono. Desde então apenas tem sido usada a implementação BlueCove que, aparte do problema já mencionado, tem funcionado correctamente.

2.3.2 Open Sound Control

O Open Sound Control não é uma biblioteca, mas sim a definição de uma especificação de como o protocolo deve ser implementado. De modo a usar este protocolo é necessário escolher um de dois caminhos. Por um lado existe a possibilidade de implementar o protocolo; por outro lado é possível usar uma das implementações disponíveis deste protocolo. Criar uma implementação em Java deste protocolo permite uma maior flexibilidade, mas esta solução foi rapidamente posta de parte, pois o tempo requerido é incompatível com os prazos definidos para o projecto.

Toona-se assim necessário recorrer a uma das implementações deste protocolo para Java. Uma pesquisa aprofundada revelou duas soluções possíveis: NetUtil [27] e JavaOSC [28] . Uma análise da API destas duas implementações mostra que são incompatíveis, ou seja, o código criado para usar uma destas implementações não pode ser usado com a outra implementação. Logo foi necessário, desde o início, escolher uma delas em exclusivo para desenvolver o DiABlu MailMan. Seguidamente é feita a análise de cada uma delas.

2.3.2.1 NetUtil

A NetUtil é uma biblioteca compacta para Java que fornece as capacidades de receber e enviar mensagens usando o protocolo Open Sound Control. Usando esta biblioteca é possível enviar mensagens OSC através dos protocolos de comunicação TCP e UDP. É uma aplicação independente da plataforma. Como principais funcionalidades tem a construção e interpretação de pacotes OSC, assim como suporte para *bundles* OSC, quer para criação como para leitura. Esta biblioteca necessita da versão 1.4 ou superior do Java e está sob uma licença LGPL.

2.3.2.2 JavaOSC

A JavaOSC é uma biblioteca Java que implementa o protocolo OSC. É bastante semelhante à biblioteca NetUtil, embora não seja possível enviar pacotes OSC usando o protocolo de comunicação TCP. Tem suporte para *bundles* OSC assim como criação e interpretação de pacotes OSC.

2.3.2.3 Conclusão

Apesar da enorme semelhança ao nível das funcionalidades fornecidas por estas duas bibliotecas, e escolha recaiu sobre a NetUtil. Em primeiro lugar porque foi a biblioteca usada nas outras aplicações do mesmo projecto, o que simplifica eventuais reutilizações de código, assim como mantém a coerência tecnológica das aplicações criadas no âmbito do projecto DiABlu. Em segundo lugar a ausência de suporte TCP na biblioteca JavaOSC é um pouco limitativo. Foi devido a estes dois

factores que foi escolhida a biblioteca NetUtil, em detrimento da biblioteca JavaOSC, para o desenvolvimento da aplicação DiABlu MailMan.

2.4 Resumo

Este capítulo foi dedicado ao estado da arte e análise tecnológica.

Na primeira parte do capítulo estudou-se uma lista de aplicações com características semelhantes, ou pelo menos vagamente parecidas, às da aplicação DiABlu MailMan. Concluiu-se que não existe nada no mercado que forneça funcionalidades Bluetooth às instalações de arte digital. Realmente, a maior parte das aplicações Bluetooth são desenvolvidas a tendo como objectivo o marketing de proximidade, pelo que garantem comunicação unidireccional.

Na segunda parte deste capítulo foram apresentadas as tecnologias escolhidas para o desenvolvimento deste projecto. Foram ainda apresentadas as possíveis alternativas, assim como uma justificação para as escolhas tomadas. Nesta parte deste capítulo foram ainda descritas as bibliotecas que foram usadas no decorrer do desenvolvimento do projecto DiABlu MailMan.

Capítulo 3

Análise e arquitectura da aplicação

Este terceiro capítulo tem como objectivo dar a conhecer melhor a aplicação DiABlu MailMan. É feita neste capítulo uma descrição mais aprofundada da aplicação e são especificados os requisitos da mesma. É ainda apresentada toda a arquitectura da aplicação.

3.1 Análise do Problema

A aplicação DiABlu MailMan tem como objectivo fornecer capacidades de transferência de ficheiros às instalações de arte digital. Admite-se que as ferramentas mais usadas na criação destas instalações têm já capacidades de comunicação por OSC.

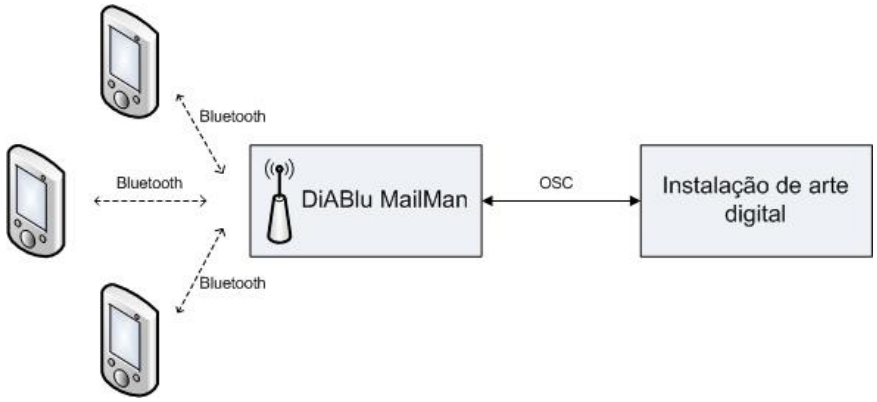

Figura 3.1 - Arquitectura de um sistema que usa o a aplicação DiABlu MailMan

Como se pode ver pela figura 3.1, a aplicação DiABlu MailMan vai situa-se entre os dispositivos móveis e a instalação de arte digital. A comunicação com os

dispositivos móveis é feita através do protocolo Bluetooth, e a comunicação com a instalação de arte digital é feita recorrendo ao protocolo OSC.

De modo a tratar das transferências por Bluetooth, é necessário separar os conceitos de envio e de recepção de ficheiros, devido às diferentes direcções que as comunicações têm em cada um deles.

O envio de ficheiros por parte da aplicação DiABlu MailMan, não é iniciado por ela. Realmente, para que a aplicação envie um ficheiro, é necessário que lhe seja comunicado um comando correspondente. Essa comunicação tem de partir da aplicação usada na criação da instalação de arte digital.

Na recepção de ficheiros, a aplicação DiABlu MailMan tem de estar permanentemente à escuta, de modo a poder receber um ficheiro, mal ele seja enviado. Após a recepção do ficheiro, é guardado e a sua localização, assim como alguma informação sobre o mesmo ficheiro são enviadas para a aplicação usada na criação da instalação de arte digital, usando o protocolo OSC.

3.1.1 Características dos Utilizadores

O DiABlu MailMan foi desenvolvido tendo sempre em conta os seus utilizadores finais. Estes são, na sua grande maioria, artistas digitais. Este tipo de utilizador geralmente não possui conhecimentos aprofundados de informática e, muito menos de protocolos de comunicação. A aplicação DiABlu MailMan suporta o protocolo OSC que, entre outras funcionalidades, permite a troca de mensagens simples e é amplamente suportado pelas ferramentas com que este tipo de utilizadores trabalha. Por recurso a este protocolo é possível o envio e recepção de ficheiros entre dispositivos móveis com capacidades Bluetooth e a instalação de arte digital, sem a necessidade do utilizador final ter de conhecer a fundo o protocolo Bluetooth.

3.1.2 Restrições

Fazendo parte do projecto DiABlu, a aplicação DiABlu MailMan tem que usar o protocolo OSC para permitir que o utilizador interaja com a aplicação. As mensagens devem ser bem definidas e em número limitado. De modo a que a aplicação comunique com os dispositivos móveis, será usado o protocolo Bluetooth, que tem sete como número máximo de comunicações simultâneas.

3.1.3 Pressupostos e Dependências

Pressupõe-se que a máquina onde irá correr a Instalação de arte digital tem como sistema operativo Windows XP com SP2 instalado ou superior, ou em alternativa Mac OS X. Pressupõe-se ainda que o utilizador final tem alguma experiência com o sistema operativo escolhido, assim como o domínio das

ferramentas usadas na criação da instalação de arte digital, que serão usadas para trocar mensagens com a aplicação DiABlu MailMan.

3.2 Especificação dos Requisitos

Nesta secção são definidos os requisitos (funcionais e não funcionais) do sistema, que garantem a correcção do desenho e da implementação. A cada um dos requisitos é atribuída uma prioridade (baixa, média ou alta) que está directamente ligada com a importância do requisito em causa. Os requisitos com prioridade alta são aqueles indispensáveis para o funcionamento do sistema; os de prioridade média não são indispensáveis, mas sem eles o sistema perde funcionalidades importantes; os requisitos com prioridade baixa são úteis, mas a funcionalidade do sistema não depende deles.

A especificação dos requisitos apresentados de seguida foi efectuada antes do início do desenvolvimento, tendo posteriormente sofrido algumas alterações, quer devido à detecção de requisitos impossíveis de cumprir devido a limitações das tecnologias usadas, quer devido à definição de novas funcionalidades.

3.2.1 Requisitos funcionais

3.2.1.1 RF1- Enviar ficheiro para um dispositivo.

Descrição: A aplicação DiABlu MailMan tem de ter a capacidade de enviar um determinado ficheiro para um determinado dispositivo. O ficheiro e o dispositivo são comunicados à aplicação DiABlu MailMan pelo protocolo OSC.

Prioridade: Alta.

3.2.1.2 RF2 - Enviar um ficheiro para um grupo.

Descrição: A aplicação DiABlu MailMan tem de ter a capacidade de enviar um determinado ficheiro para um grupo de dispositivos. Um grupo de dispositivos pode ser definido como: dispositivos de uma determinada *major device class* (computador, telefone, etc.), de uma determinada *minor device class* da *major device class escolhida* (no caso dos computador temos: laptop, desktop, servidor, pda, etc) ou de uma determinada marca (Nokia, Siemens, LG, etc.). Este envio deve ser feito para todos os dispositivos que pertençam ao grupo e que se encontrem ao alcance da aplicação DiABlu MailMan. O envio deve ser feito o mais prontamente possível.

Prioridade: Média.

3.2.1.3 RF3 - Enviar um ficheiro para todos os dispositivos.

Descrição: A aplicação DiABlu MailMan tem de ter a capacidade de enviar um determinado ficheiro para todos os dispositivos presentes ao seu alcance num determinado momento. O ficheiro a enviar é comunicado à aplicação pelo protocolo OSC. Este envio deve ser feito o mais prontamente possível

Prioridade: Alta.

3.2.1.4 RF4 - Receber um ficheiro.

Descrição: A aplicação DiABlu MailMan tem de ter a capacidade de receber um ficheiro de um dispositivo móvel com capacidade Bluetooth. O caminho para o ficheiro armazenado é memorizado; terá de ser codificado numa mensagem OSC e enviado para a instalação de arte digital, usando o protocolo OSC.

Prioridade: Alta.

3.2.1.5 RF5 - Gerar e enviar mensagens OSC.

Descrição: Quando necessário, a aplicação DiABlu MailMan tem de conseguir gerar mensagens OSC de modo a transmitir informação para a ferramenta que está a utilizar a aplicação. Estas mensagens são pré-definidas, com alguns parâmetros configuráveis e serão enviadas usando o protocolo OSC.

Prioridade: Alta.

3.2.1.6 RF6 - Receber e interpretar mensagens OSC.

Descrição: Quando a aplicação DiABlu MailMan recebe uma mensagem OSC, essa mensagem deve ser interpretada correctamente e uma acção deve ser desencadeada, dependendo da mensagem recebida.

Prioridade: Alta.

3.2.1.7 RF7 - Enviar uma resposta quando uma mensagem OSC é recebida.

Descrição: Quando uma mensagem OSC é recebida, é necessário enviar uma resposta. Essa resposta depende da acção que a mensagem desencadeou, e deve conter o resultado dessa acção.

Prioridade: Alta.

3.2.1.8 RF8 - Guardar sessão.

Descrição: A aplicação DiABlu MailMan deve gravar a sessão actual em ficheiro. Este ficheiro deve conter toda a informação necessária para que a aplicação possa ser reposta no estado em que se encontrava no memento em que a sessão foi arquivada. A sessão deve ser arquivada automaticamente em intervalos de tempo regulares e sempre que a aplicação DiABlu MailMan é encerrada. Nos dados gravados no ficheiro devem constar os dispositivos que foram encontrados na rede, os que enviaram ficheiros e quais os ficheiros enviados, os dispositivos receberam ficheiros e quais os ficheiros recebidos, e os dispositivos que rejeitaram ficheiros. Devem ainda ser guardadas as definições necessárias do sistema.

Prioridade: Média.

3.2.1.9 RF9 - Abrir sessão.

Descrição: A aplicação DiABlu MailMan deve ter a capacidade de ler um ficheiro com uma sessão arquivada anteriormente. Este ficheiro permitirá restaurar a aplicação, de modo a que fique em situação idêntica à que possuia quando foi arquivada.

Prioridade: Média.

3.2.1.10 RF10 - Painel de configuração.

Descrição: A aplicação DiABlu MailMan tem de disponibilizar um painel de configuração na sua interface gráfica. Neste painel tem de ser possível, entre outras funcionalidades, configurar o endereço IP e a porta do servidor OSC, assim como o endereço e a porta do cliente.

Prioridade: Alta.

3.2.1.11 RF10.2 - Testar envio e recepção de ficheiros

Descrição: A aplicação DiABlu MailMan deve ter um painel de teste onde é permitido testar o envio e recepção de ficheiros quando a aplicação DiABlu MailMan não está em funcionamento. Quando o teste é efectuado, não será gerada nenhuma mensagem OSC e todas as mensagens, quer de sucesso, quer de erro, serão mostradas imediatamente ao utilizador.

Prioridade: Baixa.

3.2.1.12 RF11 - Permitir que a ligação seja configurada pelo cliente.

Descrição: A aplicação DiABlu MailMan deve permitir que o cliente envie o seu endereço IP e porta à qual se deve ligar.

Prioridade: Baixa.

3.2.1.13 RF12 - Mostrar registo

Descrição: A aplicação DiABlu MailMan deve mostrar um registo de todas as comunicações efectuadas (ficheiros enviados e recebidos, mensagens OSC enviadas e recebidas, respostas), assim como o resultado dessas comunicações (ficheiro aceite, ficheiro rejeitado, comunicação falhada, etc.).

Prioridade: Média.

3.2.1.14 RF13 - Gravar registo

Descrição: A aplicação DiABlu MailMan deve guardar o registo das comunicações efectuadas, para que estas possam ser analisadas mais tarde.

Prioridade: Alta.

3.2.1.15 RF14 - Lançar a aplicação DiABlu MailMan por linha de comandos.

Descrição: Deve ser possível inicializar a aplicação por linha de comandos, não devendo nestas condições apresentar qualquer interface gráfica. Ao ser inicializada desta forma, a configuração será lida de um ficheiro.

Prioridade: Alta

3.2.2 Requisitos não funcionais

Nesta secção, são identificados e descritos os requisitos não funcionais considerados essenciais para o desenvolvimento da aplicação DiABlu MailMan.

3.2.2.1 RNF1 - Funcionalidade

RNF1.1 – Interoperabilidade

A aplicação DiABlu MailMan deve conseguir interagir de uma forma bastante simplificada com as instalações de arte digital que a irão usar. Para o efeito, é necessário que as mensagens OSC que são trocadas sejam completamente definidas, assim como as respectivas respostas.

RNF1.2 – Adequabilidade

O produto a desenvolver deverá fornecer o conjunto de funcionalidades adequadas às necessidades dos utilizadores que constam neste documento.

RNF1.3 – Segurança

Deverá ser garantida a segurança da aplicação e do sistema em que esta irá correr. Para o efeito, será necessária a opção de limitar os tipos de ficheiros que serão aceites pela aplicação.

3.2.2.2 RNF2 – Fiabilidade

O sistema deve desempenhar a sua função, sob condições específicas, de forma adequada, como previsto durante o período de desenvolvimento.

3.2.2.3 RNF3 - Usabilidade

RNF3.1 – Atractividade

Apesar de não ser extremamente necessário, torna-se menos cansativo usar uma aplicação com uma interface gráfica atractiva. Além do aspecto estético, é também necessário ter em atenção a disposição dos elementos com que o utilizador irá interagir.

RNF3.2 – Compreensibilidade

É desejável que o sistema dê um feedback e apresente mensagens esclarecedoras em caso de erro por parte do utilizador ou mesmo por falha do próprio sistema. Também é desejável que a aplicação mostre informação com o intuito de ajudar o utilizador, sempre que necessário, e que possua um manual simples e fácil de consultar, em suporte digital.

RNF3.3 - Fácil aprendizagem

Deve ser permitido aos utilizadores usarem as funções da aplicação com o mínimo esforço necessário e de uma forma intuitiva, de forma a aumentar a facilidade com que este utilizadores aprenderem a trabalhar com a aplicação.

RNF3.4 – Operabilidade

É fundamental que todo o sistema seja configurável. A configuração pode ser feita através de um ficheiro, chamada da linha de comandos, ou no painel de configuração da aplicação

3.2.2.4 RNF4 – Eficiência

É necessário que as transferências sejam feitas o mais rapidamente possível, respeitando os limites do hardware. As mensagens OSC recebidas devem ser interpretadas rapidamente, e as acções respectivas devem ser desencadeadas o mais prontamente possível.

3.2.2.5 RNF5 - Fácil Manutenção

RNF5.1 - Alterabilidade

A arquitectura do sistema deve permitir facilmente escalar o número de variáveis da aplicação. Para além disso, a arquitectura deve ser modular, para que seja possível, em qualquer altura, adicionar novas funcionalidades

RNF5.2 - Facilmente analisável

É fundamental que todo o sistema seja desenvolvido com uma arquitectura que permita, facilmente, detectar eventuais problemas e localizar facilmente as partes necessárias a modificar.

RNF5.3 - Estabilidade

É necessário que o sistema esteja organizado numa arquitectura modular em todos os seus componentes de software, através da criação de interfaces e adopção de normas e convenções de codificação, para que quando for necessário alterar ou estender o software, não se criem efeitos secundários e inesperados no funcionamento do sistema.

RNF5.4 - Cobertura de testes

Os testes serão realizados aquando do desenvolvimento do sistema. Serão, para tal, realizados de modo a que permitam demonstrar de forma sistemática que as funcionalidades disponibilizadas estão de acordo com os requisitos referidos neste documento.

3.2.2.6 RNF6 - Portabilidade

O sistema terá como sistemas operativos alvo o Windows XP com o SP2 instalado, assim como Mac OSX. O sistema poderá funcionar em Linux, mas não será garantido um correcto funcionamento.

3.2.3 Diagramas de casos de uso

Uma das ferramentas mais importantes para a boa compreensão do funcionamento de uma aplicação é o diagrama de casos de uso. Este diagramas permite compreender quais os actores que interagem com a aplicação, e quais são essas interacções.

3.2.3.1 Actores

A aplicação DiABlu MailMan foi desenvolvida com o objectivo de interagir com três actores diferentes:

- **Utilizador –** O utilizador é entendido como a pessoa que pretende usar a aplicação DiABlu MailMan, para realizar a sua instalação de arte digital.
- **Dispositivo Bluetooth** – Os dispositivos podem interagir com a aplicação, sozinhos ou em grupo. Estes grupos de dispositivos podem ser um grupo específico (por exemplo todos os telemóveis Nokia), ou todos os dispositivos ao alcance da aplicação.
- **Instalação de arte digital** – A instalação de arte digital é o actor mais importante, pois foi a pensar nele que esta aplicação foi desenvolvida. A instalação de arte digital pode ser vista como uma ou mais peças de software desenvolvidas numa ou mais aplicações, e que está dotada de funcionalidades de comunicação usando o protocolo OSC.

3.2.3.2 Casos de uso de um utilizador

Como se pode ver na figura 3.2, o utilizador tem apenas duas opções possíveis, lançar e configura a aplicação. A primeira está relacionada com o início da aplicação. A aplicação DiABlu MailMan pode ser inicializada de duas formas diferentes:

- **Com interface gráfica** – Quando inicializada desta forma, a aplicação está totalmente disponível para o utilizador. Todas as funcionalidades de configuração estão disponíveis, e todas as mensagens são mostradas na própria interface gráfica.
- **Sem interface gráfica** – Quando a aplicação é lançada sem interface gráfica deixa de ser possível ao utilizador configurar a aplicação em *runtime.* A configuração tem de ser feita antes da execução da aplicação, recorrendo à edição de um ficheiro.

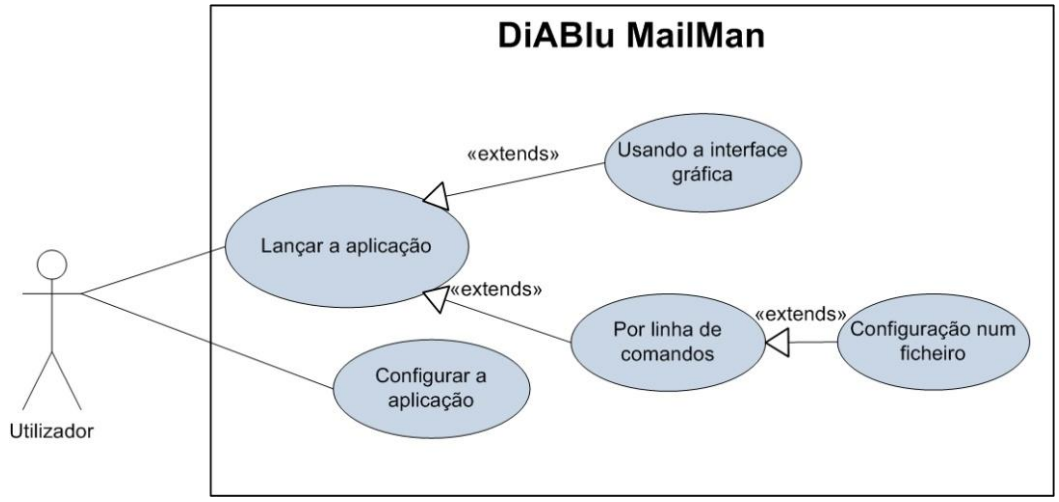

Figura 3.2 - Casos de uso do utilizador

A segunda opção do utilizador corresponde a configurar a aplicação. Esta configuração é feita de uma forma diferente, dependendo da forma como o utilizador pretende inicializar a aplicação DiABlu MailMan.

3.2.3.3 Casos de uso de Dispositivo Bluetooth

A aplicação DiABlu MailMan permite que seja feito o envio de ficheiros para um dispositivo, para um grupo de dispositivos ou para todos os dispositivos Bluetooth. A aplicação DiABlu MailMan suportará também a recepção de ficheiros enviados por um dispositivo Bluetooth.

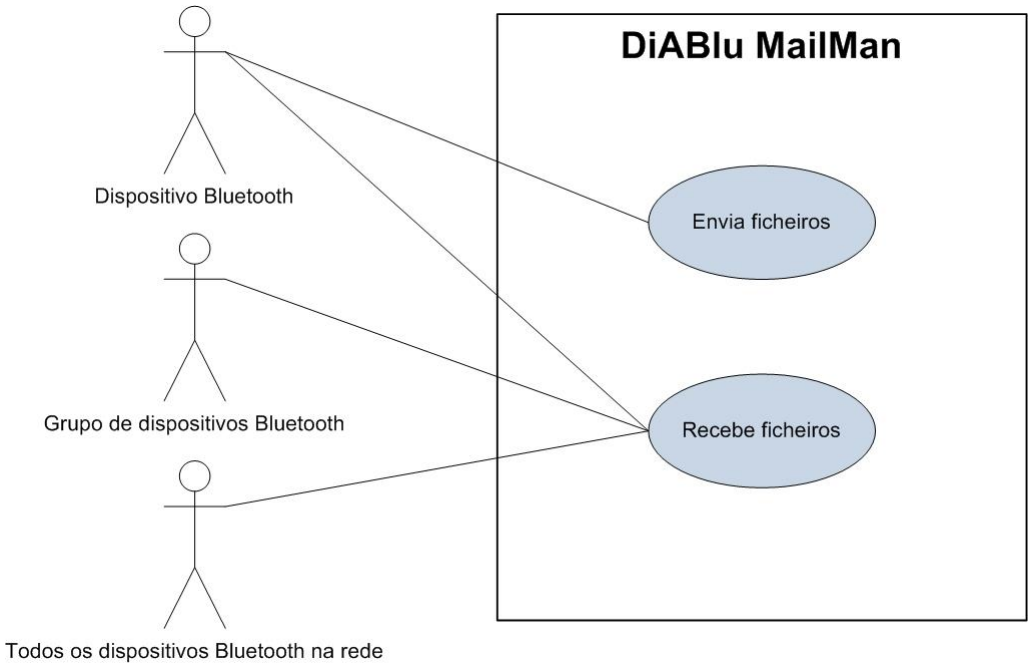

Figura 3.3 - Casos de uso dos dispositivos Bluetooth

3.2.3.4 Casos de uso da instalação de arte digital

A instalação de arte digital que usa o DiABlu MailMan necessita de enviar e de receber ficheiros de dispositivos Bluetooth. Para tal, a aplicação DiABlu MailMan fornece uma interface de comunicação OSC que disponibiliza funcionalidades adequadas

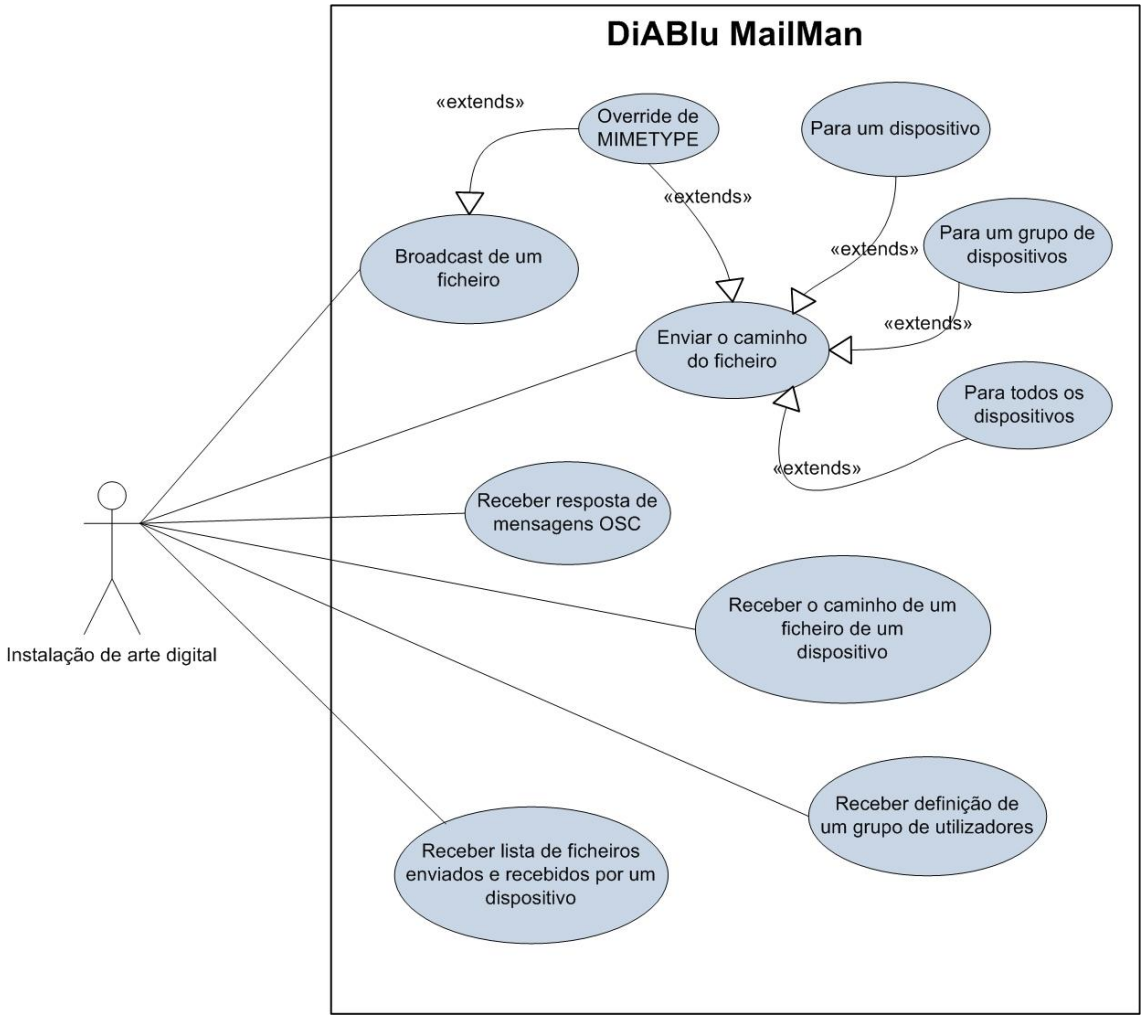

Figura 3.4 - Casos de uso da instalação de arte digital

Através dessa interface, a instalação de arte digital pode requisitar à aplicação DiABlu MailMan o envio de um ficheiro para um dispositivo Bluetooth, um grupo de dispositivos Bluetooth, ou todos os dispositivos Bluetooth disponíveis ao alcance da aplicação, indicando a localização desse ficheiro. Se necessário, a aplicação pode especificar qual o *mimetype* do ficheiro a enviar; neste caso o *mimetype* especificado será o requisitado pela instalação de arte digital em vez do que se encontra definido por omissão.

A instalação de arte digital pode ainda requisitar um *broadcast* de um ficheiro. O *broadcast* é uma funcionalidade da aplicação DiABlu MailMan que consiste em

enviar um ficheiro para todos os dispositivos que se encontrem ao alcance da aplicação DiABlu MailMan, num qualquer instante dentro de um intervalo de tempo definido.

Além dos casos de uso acima referidos, a instalação de arte digital pode requisitar alguma informação à aplicação DiABlu MailMan. A informação a que a instalação tem acesso é:

- A lista de dispositivos que pertencem a um determinado grupo
- A lista de ficheiros enviados para um dispositivo
- A lista de ficheiros recebidos de um dispositivo

Todos os casos de uso apresentados são realizados recorrendo ao protocolo OSC. Cada pedido à aplicação DiABlu MailMan traduz-se numa mensagem OSC enviada pela instalação de arte digital. Cada mensagem OSC enviada pela instalação de arte digital produz uma mensagem OSC de resposta por parte da aplicação DiABlu MailMan. Essa mensagem OSC de resposta depende do resultado das acções desencadeadas pelo pedido feito pela instalação de arte digital. Pode ser uma mensagem de sucesso, de erro, ou uma mensagem com a informação requisitada.

3.3 Arquitectura do DiABlu MailMan

3.3.1 Estrutura

O estudo dos requisitos do sistema revelou que é vantajoso usar, no desenvolvimento desta aplicação, uma estrutura modular. Usar módulos permite agrupar funcionalidades que estão relacionadas entre si, o que facilita consideravelmente o desenvolvimento da aplicação.

Uma análise dos requisitos leva a que se separe a aplicação em quatro grandes módulos. E são eles:

- Módulo Bluetooth
- Módulo OSC
- Módulo de gestão de eventos
- Módulo de configuração

3.3.1.1 Módulo Bluetooth

Este módulo é responsável por todas as acções relacionadas com o protocolo de comunicação Bluetooth. Estas acções estão divididas em duas grandes partes: envio e recepção de ficheiros e descoberta de dispositivos.

A primeira destas duas partes é responsável por tratar de todos os envios dos ficheiros, assim como preparar a aplicação de modo a que esta possa receber ficheiros através do protocolo Bluetooth.

A parte de pesquisa de dispositivos é responsável por, quando necessário, procurar todos os dispositivos ao alcance da aplicação DiABlu MailMan. Esta pesquisa, além de encontrar dispositivos, determina quais se esses dispositivos têm activos os serviços necessários à transferência de ficheiros.

3.3.1.2 Módulo OSC

O módulo OSC é responsável por toda a comunicação da aplicação com a instalação de arte digital. Este módulo contém as funcionalidades necessárias para criar um servidor OSC e um cliente OSC, ambos necessários para a criação de uma comunicação nos dois sentidos. O módulo OSC disponibiliza ainda funcionalidades para criação, interpretação, envio e recepção de mensagens OSC.

3.3.1.3 Módulo de gestão de eventos

Este é o módulo principal da aplicação e destina-se a estabelecer a comunicação restantes os outros módulos. Este módulo é principalmente passivo, ou seja, apenas age quando solicitado pelos restantes. A única excepção é no início da aplicação, altura em que realiza todas as inicializações necessárias ao bom funcionamento da aplicação.

3.3.1.4 Módulo de configuração

No módulo de configuração estão presentes todas as funcionalidades que permitem a configuração da aplicação DiABlu MailMan. Este módulo permite definir as propriedades da ligação OSC (endereço e porta da máquina anfitriã a que se deve ligar), assim como escolher o modo de visualização dos *logs*.

3.3.2 Arquitectura lógica

3.3.2.1 Vista Horizontal

A aplicação está decomposta em três camadas distintas: a camada de interface, a camada de lógica de negócios e a camada de dados. A camada de interface é a que proporciona a interacção entre a aplicação e os elementos externos e encontra-se subdividida em três partes distintas:

- **Interface Bluetooth** - através da qual os dispositivos móveis vão interagir com a Aplicação DiABlu MailMan;
- **Interface OSC** - através da qual a Instalação de arte digital vai interagir com a Aplicação DiABlu MailMan;

- **Interface de configuração -** através da qual o utilizador final pode alterar algumas definições do sistema.

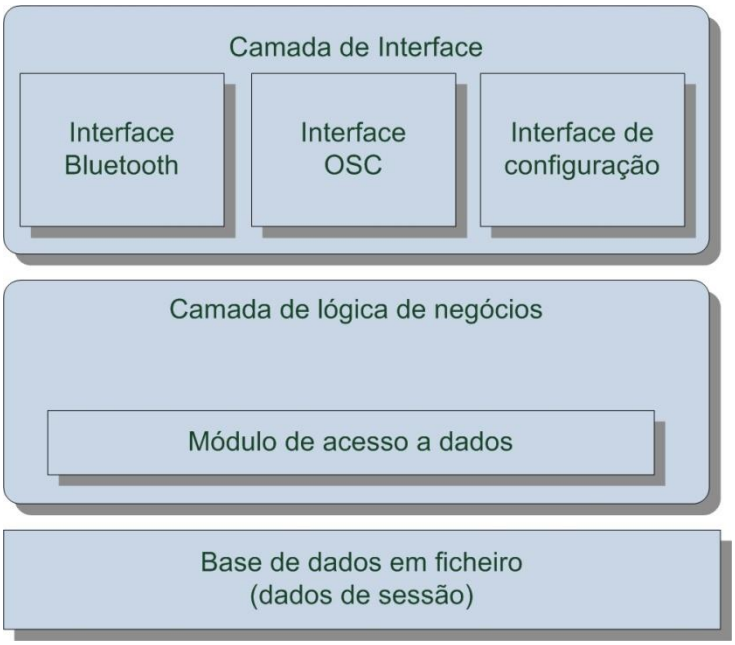

Figura 3.5 - Vista horizontal da aplicação

Na camada de lógica de negócios são tratados todos os resultados das interacções com as interfaces da camada superior e todos os eventos consequentes são desencadeados. Esta camada é a parte mais complexa da aplicação e, consequentemente, a mais pesada computacionalmente. Esta camada contém ainda o módulo de acesso a dados que permite estabelecer comunicação com a camada inferior, a camada de base de dados.

A camada de base de dados é a terceira e última camada e consiste num conjunto de ficheiros onde se pode guardar sessões e configurações. O facto de serem guardadas as sessões permite que elas possam ser repostas posteriormente. Deste modo garante-se uma maior robustez do sistema, assim como uma maior tolerância a falhas. Nesta camada são ainda gardadas todas as definições da aplicação, de modo a que as definições configuradas no ultimo uso da aplicação sejam mantidas.

3.3.2.2 Vista Vertical

A decomposição vertical é feita por subsistemas, de uma forma hierárquica, em que cada sistema corresponde a um grupo de funcionalidades que pode abranger todas as camadas da implementação. Apresenta-se de seguida o modelo mais funcional da arquitectura do sistema com base num diagrama de pacotes vertical, que permite visualizar as interacções entre os diversos componentes do sistema.

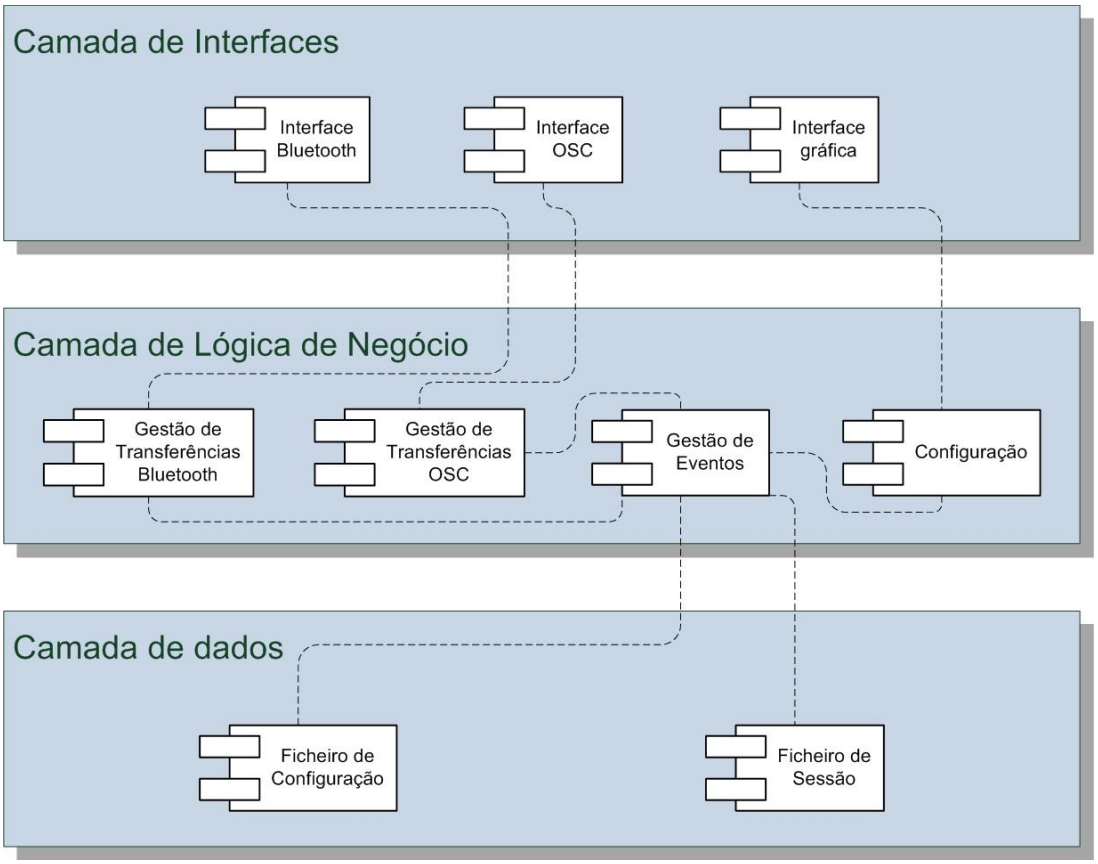

Figura 3.6 - Vista vertical da aplicação

Na camada de interfaces existem três componentes distintos. Os dois primeiros correspondem às interfaces de comunicação (Bluetooth e OSC), e o terceiro a uma interface gráfica que permite, ao utilizador, configurar a aplicação. Estes componentes podem ser vistos como as ligações que a aplicação tem com o exterior.

A camada da lógica de negócio pode ser entendida como sendo o "cérebro" da aplicação. Esta camada é composta por quatro componentes:

- **Componente de configuração -** está responsável por tratar da configuração do sistema, e garantir a continuação de um bom funcionamento quando as definições são alteradas.
- **Componente de gestão de transferências Bluetooth -** trata de toda a lógica associada à transferência de ficheiros usando dispositivos Bluetooth.
- **Componente de gestão de transferências OSC -** trata de toda a lógica associada à transferência de ficheiros usando o protocolo OSC. Estão

incluídos neste componente as funcionalidades de criação e interpretação de mensagens OSC.

- **Componente de gestão de eventos** - Este é o componente mais importante da aplicação; trata da gestão de tudo o que se passa nos outros componentes da mesma camada e sempre que algo acontece noutro componente, toma conhecimento e desencadea as acções necessárias.

Por último, temos a camada de dados, que conta com apenas dois componentes. O primeiro é um ficheiro de configuração, que contém as definições do sistema e que permite que, da vez seguinte que a aplicação seja inicializada, as definições se mantenham as mesmas. O segundo componente é o ficheiro de sessão, que contém toda a informação necessária para restaurar uma sessão anterior. Este ficheiro garante uma maior robustez e tolerância a falhas da aplicação.

3.3.2.3 Arquitectura física

A arquitectura física permite visualizar a topologia dos diversos componentes de hardware e de software do sistema. É ainda possível ver numa arquitectura física bem definida as ligações entre esses mesmos componentes. O diagrama da figura X foi elaborado para permitir uma mais fácil visualização da estrutura física de alto nível da aplicação DiABlu MailMan.

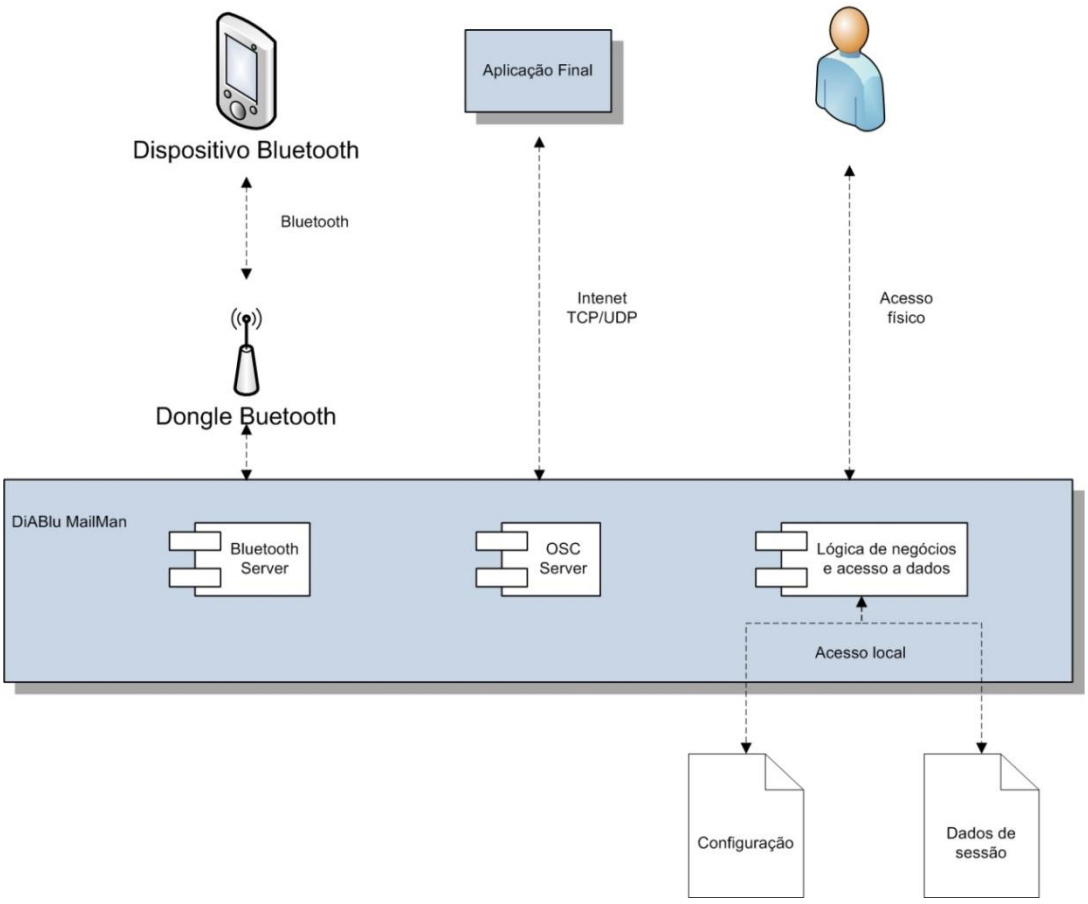

Figura 3.7 - Arquitectura física da aplicação

No diagrama da figura X, é possível ver a arquitectura física de um sistema em funcionamento, em que a instalação de arte digital utiliza a aplicação DiABlu MailMan. No topo do diagrama da figura X estão representados todos os intervenientes neste sistema:

- Um dispositivo móvel com capacidades Bluetooth, comunicando com um *dongle* Bluetooth. Esta ligação é posteriormente tratada pela aplicação DiABlu MailMan.
- A instalação de arte digital. De modo a interagir com o sistema, esta instalação deverá estar dotada de capacidades de comunicação através do protocolo OSC. Usando este protocolo, comunica com a aplicação DiABlu MailMan através da internet ou de uma rede de computadores, usando o protocolo UDP.
- Utilizador que pode interagir directamente com a aplicação DiABlu MailMan, com o intuito de a configurar, necessitando para isso de acesso físico à máquina onde esta aplicação está a ser executada

Dentro da aplicação DiABlu MailMan existem três componentes principais: dois servidores, o de Bluetooth e o de OSC, assim como o componente de lógica e acesso a dados. Os dois primeiros tratam das respectivas comunicações enquanto que o componente de lógica de negócios e acesso a dados, trata de todos os eventos gerados pelos outros componentes da aplicação e comunica-lhes as acções que devem concretizar. Além disso, este componente ainda vai tratar de toda a leitura e escrita de dados em ficheiros.

Por último temos os dois ficheiros, o de configuração e o de dados de sessão. Estes ficheiros vão ser acedidos pelo componente de lógica de negócios e acesso a dados, pelo que ambos s devem estar na mesma máquina que a aplicação DiABlu MailMan.

3.4 Definição das Mensagens OSC

Esta secção é dedicada à definição das mensagens OSC, que são trocadas entre a Aplicação DiABlu Mailman e a Instalação de arte digital. Todas as mensagens têm o prefixo, “/Diablu/Mailman”, seguido do nome do comando e dos seus argumentos. O prefixo das mensagens foi escolhido para de forma a manter a consistência com outras aplicações do projecto DiABlu.

3.4.1 Mensagens de envio

3.4.1.1 SendPath

Mensagem: /DiaBlu/MailMan/SendPath”

Argumentos:

- caminho do ficheiro
- dispositivo(s)

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Envia o ficheiro que se encontra no caminho indicado na mensagem para um ou vários dispositivos. Cada dispositivo é identificado pelo seu UUID.

Resposta: “OK” ou “SendFailed”.

3.4.1.2 SendPathWithMime

Mensagem: “/Diablu/Mailman/SendPathWithMime”

Argumentos:

- caminho do ficheiro
- mimeType
- dispositivos(s)

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Envia o ficheiro que se encontra no caminho indicado na mensagem para um ou vários dispositivos, com um MimeType definido pela Instalação de arte digital. Cada dispositivo é identificado pelo seu UUID.

Resposta: “OK”, “SendFailed”.

3.4.1.3 SendPathToGroup

Mensagem: “/DiaBlu/MailMan/SendPathToGroup”

Argumentos:

- caminho do ficheiro
- major device class
- minor device class
- marca

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Envia o ficheiro que se encontra no caminho indicado na mensagem para um ou vários dispositivos definidos pelo grupo indicado na mensagem. É permitido o uso da wildcard '*' na definição de grupo.

Resposta: "OK", "SendFailed".

3.4.1.4 3.8 SendPathWithMimeToGroup

Mensagem: "/Diablu/Mailman/SendPathWithMimeToGroup"

- caminho do ficheiro
- mimeType
- major device class
- minor device class
- marca

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Envia o ficheiro que se encontra no caminho indicado na mensagem para um ou vários dispositivos definidos pelo grupo indicado na mensagem, com um MimeType definido pela Instalação de arte digital. É permitido o uso da wildcard '*' na definição de grupo.

Resposta: "OK", "SendFailed".

3.4.1.5 BroadcastPath

Mensagem: "/DiaBlu/Mailman/BroadcastPath"

- caminho do ficheiro
- tempo de broadcast
- tempo entre pesquisa de dispositivos
- major device class
- minor device class
- marca

Sentido: DiABlu MailMan → Instalação de arte digital.

Descrição: Envia um ficheiro que se encontra no caminho indicado na mensagem para todos os dispositivos encontrados durante um tempo definido na mensagem. Caso o tempo seja '0', o ficheiro só será enviado para os dispositivos presentes no momento. É necessário definir o intervalo de tempo entre pesquisas de dispositivos.

Resposta: "OK", "SendFailed".

3.4.2 Mensagens de recepção

3.4.2.1 ReceivePath

Mensagem: "/DiaBlu/Mailman/ReceivePath"

Argumentos:

- caminho do ficheiro
- dispositivo

Sentido: DiABlu MailMan → Instalação de arte digital.

Descrição: Envia o caminho de um ficheiro recebido por um dispositivo para a Instalação de arte digital. O dispositivo é identificado pelo seu UUID

Resposta: Não há resposta.

3.4.3 Mensagens de informação

3.4.3.1 GetGroup

Mensagem: "/Diablu/Mailman/GetGroup"

Argumentos:

- MajorClass
- MinorClass
- marca

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Pede a lista de dispositivos que pertencem a um determinado grupo. O grupo pode ser definido pela *major device class*, *minor device class* ou pela marca (Nokia, Siemens, etc.). O preenchimento dos três campos é obrigatório, embora a wildcard '*' possa ser usada. A marca do dispositivo é obtida a partir da lista de endereços atribuídos a cada fabricante. Apesar de não ser completamente infalível, é a única forma de obter essa informação de um dispositivo Bluetooth.

Resposta: "GroupDefinition".

3.4.3.2 GetFileList

Mensagem: "/Diablu/Mailman/GetFileList"

Argumentos:

- dispositivo

Sentido: Instalação de arte digital → DiABlu MailMan.

Descrição: Pede a lista de ficheiros enviados e recebidos por um dispositivo. O dispositivo é identificado a partir do seu UUID

Resposta: "FileListResponse".

3.4.4 Mensagens de resposta

3.4.4.1 OK

Mensagem: "/Diablo/Mailman/OK"

Sentido: DiABlu MailMan → Instalação de arte digital.

Descrição: Confirma o sucesso do envio de um ficheiro.

Resposta: Não há resposta.

3.4.4.2 SendFailed

Mensagem: "/Diablu/Mailman/SendFailed"

Argumentos:

- Ficheiro
- dispositivo(s)

Sentido: DiABlu MailMan → Instalação de arte digital

Descrição: Indica a lista de dispositivos para o qual falhou o envio de um ficheiro.

Resposta: Não há resposta.

3.4.4.3 GroupDefinition

Mensagem: "/Diablu/Mailman/Groupdefinition"

Argumentos:

- MajorClas
- MinorClass
- Marca
- Dispositivo(s)

Sentido: DiABlu MailMan → Instalação de arte digital.

Descrição Indica a lista de dispositivos que pertencem a um determinado grupo.

Resposta: Não há resposta.

3.4.4.4 FileListResponse

Mensagem: "/Diablu/Mailman/FileListResponse"

Argumentos:

Dispositivo

- nº de ficherios enviados
- nº de ficheiros recebidos
- ficheiros enviados
- ficheiros recebidos

Sentido: DiABlu MailMan → Instalação de arte digital.

Descrição: Indica a lista de ficheiros enviados e recebidos por um determinado dispositivo.

Resposta: Não há resposta.

3.5 Resumo

Este capítulo foi dividido em quatro partes. Na primeira parte foi exposta a especificação de requisitos da aplicação DiABlu MailMan que definem as funcionalidades que a aplicação desenvolvida deve fornecer. Na segunda parte do capítulo foram apresentados os casos de uso da aplicação, que demonstram como é que se processa a interacção. Na terceira parte foi explicada a arquitectura da aplicação. Em primeiro lugar mostra-se a estrutura modular, seguida das vistas horizontal e vertical da arquitectura. Finalmente é feita uma descrição dos comandos OSC que a aplicação DiABlu MailMan pode receber, ou enviar à instalação de arte digital.

A aplicação DiABlu MailMan encontra-se assim completamente especificado, podendo passar-se, no capítulo seguinte, aos detalhes de implementação.

Capítulo 4

Detalhes da implementação

Neste capítulo são apresentadas as partes mais relevantes da implementação da aplicação DiABlu MailMan, que estão directamente relacionadas com os requisitos e arquitectura expostos no capítulo anterior

4.1 Bluetooth

O pacote de Bluetooth está dividido em dois pacotes: comunicações e descoberta de dispositivos e serviços. O pacote de comunicações é a responsável pelo envio e recepção de ficheiros e está dividido em três classes

- BTFileReciever
- BTRequestHandler
- BTFileSender

O pacote de descoberta de dispositivos e serviços permite detectar dispositivos que estejam ao alcance da aplicação DiABlu MailMan, e quais os serviços activos em cada um.

4.1.1 Comunicações

4.1.1.1 BTFileReciever

Esta classe é a responsável por criar as condições necessárias para que a aplicação possa ser detectada por outros dispositivos, e para que esta esteja apta a receber ficheiros.

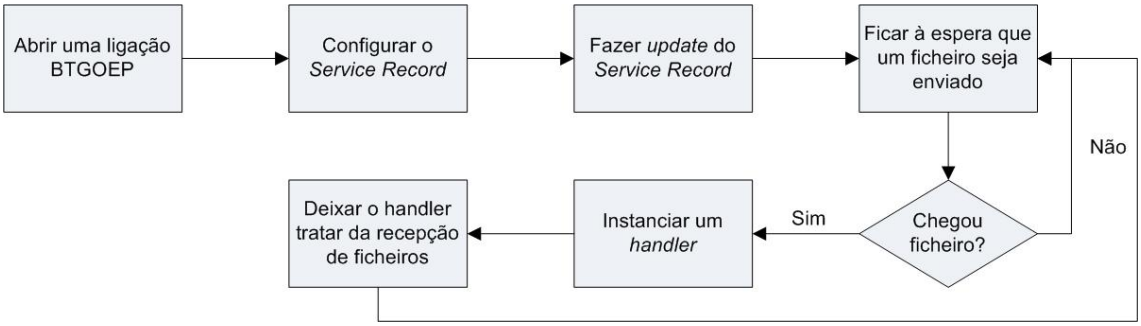

Figura 4.1 - Fluxograma da classe BTFileReceiver

Como se pode ver no fluxograma da figura X, a classe BTFileReceiver não é a responsável pela recepção de ficheiros. Essa funcionalidade é da responsabilidade da classe BTRequestHandler.

No primeiro bloco do fluxograma abre-se uma ligação usando o perfil BTGOEP (BlueTooth Generic Object Exchange Profile). É este perfil que permite o uso do protocolo OBEX. De uma forma simplificada, este bloco cria um servidor OBEX na máquina anfitrião, que só aceita ligações Bluetooth.

O passo seguinte do fluxograma da figura X é configurar o *Service Record*. Como já foi explicado no segundo capítulo, o *Service Record* contém toda a informação sobre os serviços que um dispositivo disponibiliza. Este *Service Record* tem de ser configurado para que o serviço que é disponibilizado pela aplicação DiABlu MailMan seja o pretendido.

Assim que o serviço está configurado é necessário fazer *update* do *Service Record*, de modo a que as alterações sejam gravadas.

Neste momento, o serviço está correctamente configurado e à espera que lhe sejam enviados ficheiros. Assim que um ficheiro chega, é instanciado um *handler* que trata da transferência desse mesmo ficheiro. Assim que o *handler* está inicie a sua função, é-lhe passado o endereço Bluetooth do dispositivo, de forma a que se possa guardar quais dispositivos é que receberam quais ficheiros.

O *handler* fica a executar numa *thread* separada, pelo que é possível voltar a ficar à espera de um novo ficheiro. Isto garante que qualquer ficheiro enviado vai ser recebido pela aplicação, pois esta vai estar sempre disponível.

4.1.1.2 BTRequestHandler

Como se pode ver na figura X, o *thread* desta classe vai ser chamada sempre que é detectada a recepção de um ficheiro. Esta classe, em concordância com o que foi dito anteriormente, trata de toda a transferência do ficheiro.

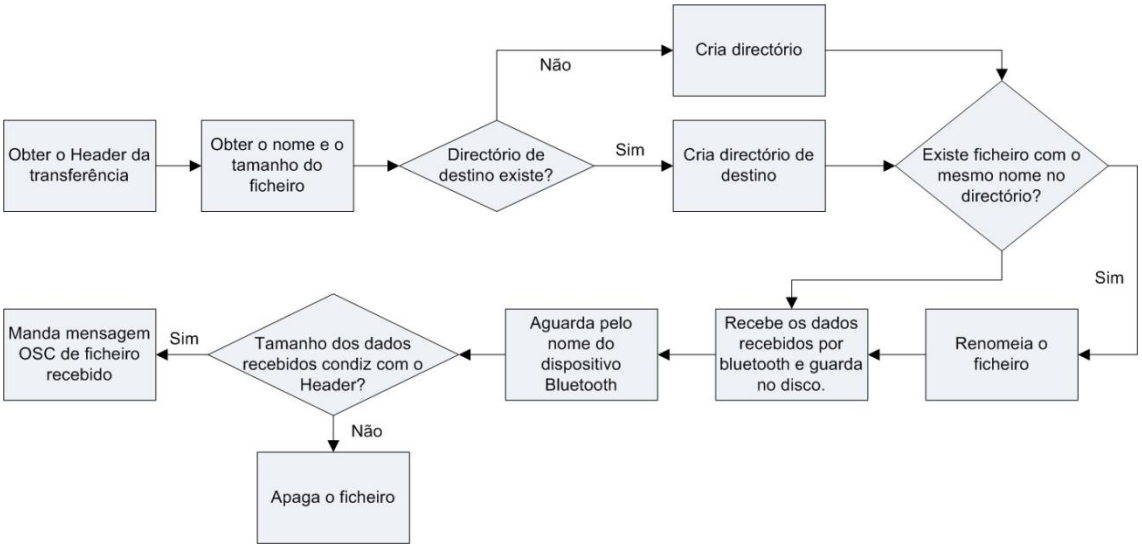

Figura 4.2 - Flowchart da class BTRequestHandler

Este *thread* é executado quando é detectada uma tentativa de envio de ficheiro para a aplicação DiABlu MailMan. O primeiro passo do algoritmo consiste em obter o *Header* da transferência. O *Header* é uma estrutura de dados que contém alguma informação sobre a transferência, da qual tem especial interesse o tamanho, o nome do ficheiro e o seu *mimetype*. Assim que esses dados são retirados do *header*, começa o processo de escrita do ficheiro.

A aplicação DiABlu MailMan permite configurar o caminho onde vai ser guardado o ficheiro. Como tal, o caminho definido pelo utilizador pode não existir, caso que leva à criação do respectivo directório. No caso de o utilizador escolher um directório inválido, a aplicação apresenta ao utilizador uma mensagem de erro, e o processo é interrompido.

Tendo agora a certeza de que o directório é válido, existe outro obstáculo a ultrapassar, que ocorre quando já existe no directório um ficheiro com o mesmo nome que o ficheiro recebido. Neste caso, é necessário mudar o nome ao ficheiro recebido. De modo a que este processo possa ser automático, foi adoptada uma convenção para a mudança de nome. O novo nome do ficheiro será igual ao nome antigo acrescentado de um número. Por exemplo se for recebido o ficheiro "exemplo.txt" e não existir nenhum outro ficheiro no directório com esse nome, o nome será mantido. No caso de existir, será guardado como "exemplo1.txt". Se o ficheiro "exemplo.txt" voltar a ser enviado, será agora guardado com "exemplo2.txt", e assim sucessivamente.

Assim que um nome válido (ou seja, um nome de um ficheiro que ainda não existe no directório) é escolhido, começa a escrita dos dados. Os dados recebidos de uma ligação Bluetooth são uma *stream* de dados, pelo que se recorreu a uma classe de Java, a InputStream, para tratar da leitura de dados. Os dados são lidos por esta

classe e escritos no ficheiro correspondente. Assim que deixa de haver dados para escrever, é necessário aguardar pelo endereço Bluetooth do dispositivo que está a enviar dados. Não havendo maneira de obter esta informação dentro desta classe, é necessário delegar esse trabalho à classe BTRecieveFile.

Estando essa fase concluída, é necessário testar a coerência dos dados. Existem vários casos em que o ficheiro chega corrompido. Por exemplo se o utilizador que está a enviar o ficheiro decide cancelar o envio, ou se sai do raio de alcance da aplicação DiABlu MailMan, o ficheiro recebido fica incompleto. É então necessário comparar o tamanho do ficheiro recebido com o tamanho indicado no *Header*. Se estes dois valores não estiverem em concordância, é concluído que o ficheiro está corrompido, e uma mensagem de erro será enviada para o utilizador.

Um ficheiro corrompido pode causar diversos problemas à instalação de arte digital. Um caso específico é o dos ficheiros JPEG. Os JPEG são delimitados por um Start OF Image (SOI) marker e um End Of Image (EOI) marker. Se o ficheiro estiver corrompido, o SOI vais estar presente mas o EOI não. Nesse caso, quando se tentar abrir o ficheiro, toda a memória alocada após o SOI vai ser entendida como parte da imagem, o que pode trazer consequências inesperadas. De forma a evitar estes problemas decidiu-se apagar todos os ficheiros corruptos.

Quando o ficheiro recebido passa este teste, uma mensagem de sucesso é enviada para o utilizador. É ainda enviada uma mensagem OSC a informar que foi recebido um ficheiro. É assim terminado o processo de recepção de um ficheiro por Bluetooth.

4.1.1.3 BTFileSender

Esta classe tem o objectivo oposto ao das duas classes expostas anteriormente. O seu objectivo é disponibilizar as funcionalidades necessárias para enviar ficheiros. O envio de ficheiros é feito de uma forma bastante mais simples do que a recepção, e pode ser feito de duas formas diferentes: com um *mimetype* padrão, ou com um *mimetype* específico. Esta funcionalidade tem utilidade quando o mimetype pré-definido do ficheiro a ser enviado não corresponde à realidade. Assim é possível escolher o *mimetype* que será enviado no Header da transmissão. O uso das funcionalidades desta classe tem algumas restrições. Para usar esta classe é necessário inserir o dispositivo na lista de dispositivos conhecidos (esta lista será explicada no decorrer deste capítulo) ou, caso ele já esteja lá presente, actualizá-lo. Passa-se então a explicar o funcionamento do envio de um ficheiro.

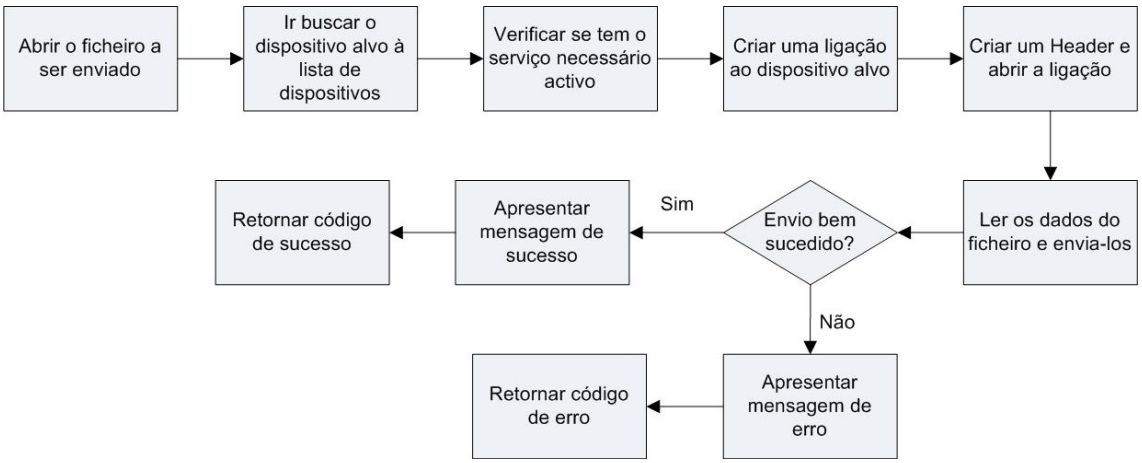

Figura 4.3 - Flowchart da classe BTFileSender

O envio de um ficheiro começa com a abertura desse mesmo ficheiro conforme pode ser visto na figura X. Assim que o ficheiro esteja aberto é necessário ir recupera uma instância do dispositivo alvo (o dispositivo para onde vai ser enviado o ficheiro) à lista de dispositivos conhecidos.

O próximo passo é verificar se o serviço de OBEX Object Push está presente no dispositivo, pois é este serviço que permite que um ficheiro seja enviado para um dispositivo Bluetooth. Este passo pode falhar por dois motivos: o dispositivo não tem o serviço activo, ou não foi inserido na lista de dispositivos conhecidos.

Se o dispositivo existe na lista e dispositivos conhecidos e tem o serviço necessário activo, o passo seguinte é criar uma ligação a esse dispositivo.

Seguidamente é criado o Header do ficheiro. Este Header é composto por: nome do ficheiro, tamanho do ficheiro e *mimetype* do ficheiro. O *mimetype* enviado depende do pretendido pelo utilizador ou, neste caso, a instalação de arte digital. Se for adoptado o uso de um *mimetype* pré definido, este será escolhido de uma lista, e depende somente da extensão do ficheiro a enviar. Por outro lado é possível especificar o *mimetype* pretendido, e será este o que será inserido no Header.

Assim que o Header estiver criado, é altura de abrir a ligação. Com a ligação aberta chega o momento de enviar os dados que são lidos do ficheiro que foi previamente aberto, e vão ser enviados para o dispositivo pretendido.

Se o envio foi bem sucedido, será apresentada uma mensagem de sucesso, e o procedimento termina, retornando um código de sucesso ao procedimento que o invocou. Se o envio não foi bem sucedido, será apresentada uma mensagem de erro e o procedimento termina com um código de erro.

4.1.2 Descoberta de dispositivos e serviços

Este pacote foi implementado de modo a que seja possível pesquisar dispositivos Bluetooth que se encontrem ao alcance da aplicação DiABlu MailMan. Alem de descobrir dispositivos, este pacote ainda disponibiliza as funcionalidades necessárias para pesquisar os serviços disponíveis num determinado dispositivo. Como foi já descrito no segundo capítulo deste documento, a pesquisa de serviços pode ser feita de uma forma generalizada, ou pode ser pesquisado um serviço específico. No caso da aplicação DiABlu MailMan, apenas interessa pesquisar o serviço OBEX Object Push. Como o tempo necessário para pesquisar um serviço é significativamente inferior ao tempo necessário para encontrar todos os serviços disponíveis num dispositivo, optou-se por se fazer uma pesquisa apenas ao serviço OBEX Object Push.

4.1.2.1 Discovery

A aplicação DiABlu MailMan tem apenas uma instância desta classe. Esta que disponibiliza duas funcionalidades: pesquisar quais os dispositivos ao alcance da aplicação DiABlu MailMan, e procurar um serviço num dispositivo.

A instanciação desta classe tem duas consequências. É instanciado um DiscoveryListner, cujo objectivo é descrito na secção seguinte, e é instanciado um agente, que é responsável por efectuar as pesquisas. Este agente é uma funcionalidade da biblioteca BlueCove, que foi usada no desenvolvimento da aplicação. Como tal, o seu funcionamento não será descrito.

Assim que esta classe esteja instanciada, é possível usar as suas duas funções:

- **Procurar dispositivos** – Quando esta funcionalidade é usada, é inicializada uma fase de inquérito (*iquery)*, realizada pelo agente. O funcionamento do processo dos inquéritos foi descrito no segundo capítulo. A partir do momento em que se requisita esta pesquisa, todo o tratamento dos resultados será efectuado pelo DiscoveryListner. Esta funcionalidade só deve ser acedida pela classe GroupGetter. No decorrer do capítulo será apresentada esta classe, assim como a razão para esta restrição.
- **Procurar um serviço** – Esta procura consiste em descobrir se um determinado serviço está activo num dispositivo remoto. O serviço em questão é o serviço OBEX Object Push. Mais uma vez todo o tratamento de resultados é feito pelo DiscoveryListner.

É de salientar que apenas uma pesquisa pode ocorrer em simultâneo. No caso de serem requisitadas várias pesquisas em simultâneo, a aplicação pode ter resultados

inesperados. Para prevenir que tal suceda está implementada uma fila de espera de pesquisas.

4.1.2.2 Discovery Listner

Esta classe está responsável por tratar dos resultados da pesquisa. Quando uma pesquisa é efectuada por uma agente, quer seja de dispositivos ou de serviços, ocorrerão eventos. Esta classe tem a capacidade de detectar esses eventos e, dependendo do evento detectado, de tomar a acção necessária.

Existem quatro eventos diferentes que podem ser detectados por esta classe. Dois dos eventos estão relacionados com a pesquisa de dispositivos, e os outros dois estão ligados à procura de serviços num dispositivo.

Dos eventos relacionados com a pesquisa de dispositivos tem-se:

- **Dispositivo descoberto** - Este procedimento é executado sempre que, durante uma pesquisa, um dispositivo é descoberto. Quando isto acontece, esse dispositivo é inserido na lista de dispositivos conhecidos e é comunicado à instância da classe GroupGetter que iniciou a pesquisa. Em resultado, no final da pesquisa, esta instância possui a lista de dispositivos presentes no raio de alcance da aplicação DiABlu MailMan
- **Fim de pesquisa de dispositivos** – Este procedimento é executado quando a pesquisa de dispositivos é terminada. Neste procedimento apenas se liberta o uso das pesquisas, ou seja, informa-se a classe Discovery que a pesquisa terminou e que, em consequencia uma próxima pode ser iniciada.

Dos eventos relacionados com a procura de serviços tem-se:

- **Serviço encontrado** – Este procedimento é executado sempre que é encontrado o serviço que estava a ser procurado num dispositivo remoto. Quando o serviço é encontrado é possível ter acesso ao seu *Service Record* que contém o URL ao qual se deve aceder para usar o serviço. Tenta-se estabelecer uma ligação a este URL e se a ligação for bem sucedida, significa que o serviço está activo e funcional no dispositivo remoto. Tal facto é guardado na instância do dispositivo presente na lista de dispositivos conhecidos. Se a ligação não for bem sucedida, admite-se que o serviço não está disponível. Esta informação é também guardada de forma semelhante na instância do dispositivo que se encontra na lista de dispositivos conhecidos.
- **Procura de serviços completa** – Este procedimento é executado de forma idêntica ao que sucede quando termina uma pesquisa de dispositivos. São accionados os dispositivos que garantiam que nenhuma

pesquisa seria feita enquanto esta estava em curso de modo a que a próxima pesquisa possa ser efectuada.

4.2 OSC

O pacote OSC é responsável pela gestão das comunicações por OSC. É ele que gere e trata todas as mensagens recebidas, desencadeando as acções correspondentes, assim como constrói as mensagens OSC a enviar. Este pacote está dividido em dois outros pacotes: um de comunicações e outro de comandos OSC

4.2.1 Comunicações

Este pacote contém três classes:

- OSCServer
- OSCClient
- OSCListener

As duas primeiras classes desta lista estão ligadas à criação e manutenção das ligações OSC, quer de entrada, quer de saída. Estas ligações vão funcionar sobre o protocolo UDP.

4.2.1.1 OSCServer

A classe OSCServer contém as funcionalidades de criar um servidor OSC de modo a que possam ser recebidas mensagens OSC. Este servidor vai ser criado na máquina local numa porte que pode ser configurada pelo utilizador. Esta classe tira proveito da biblioteca NetUtil, tornando transparente o uso do protocolo de comunicações para o programador.

Esta classe fornece apenas duas funcionalidades: iniciar e terminar o servidor.

Quando é iniciado o servidor, é aberta a porta definida pelo utilizador, através da qual é possível receber mensagens OSC. De modo a que a recepção destas mensagens OSC seja detectada, é também inicializada uma instância da classe OSCListener.

Quando a aplicação DiABlu MailMan é executada, o servidor é criado e iniciado automaticamente. O servidor pode ser terminado a qualquer momento, a pedido do utilizador.

4.2.1.2 OSCClient

A classe OSCClient faculta as funcionalidades para a criação de um cliente OSC. Quando a classe é instanciada, é criado um cliente configurado para se ligar a uma máquina cujo endereço IP e porta foram previamente definidos pelo utilizador.

Visto que o protocolo de comunicação usado para transmissão de mensagens OSC é o protocolo UDP, não faz sentido manter uma ligação aberta. O cliente só se vai ligar a um servidor OSC quando for requisitado o envio de uma mensagem. Deste modo, permite-se que um servidor OSC possa receber mensagens de vários clientes diferentes.

4.2.1.3 OSCListener

Esta classe está responsável por verificar se chegou alguma mensagem ao servidor OSC; Quando uma mensagem chega, passa a ser interpretada. O primeiro passo é saber qual o comando OSC que chegou. Caso não seja um comando válido, será enviada uma mensagem OSC de erro de volta para a aplicação que a enviou.

Se a mensagem OSC é válida, o passo seguinte é identificar e validar os argumentos. Neste passo verifica-se se o número de argumentos está em concordância com o número de argumentos esperados para o comando especificado na mensagem. Caso o número de argumentos seja válido é então verificado o tipo de dados dos argumentos.

Se a mensagem OSC passar os passos todos de validação, é então invocado o método correspondente à mensagem OSC que chegou. Após a invocação desse método, é enviada uma mensagem OSC com o resultado da operação. É enviada uma mensagem de sucesso ou uma mensagem de erro, neste caso com informação sobre o erro que ocorreu.

4.2.2 Comandos OSC

Os comandos OSC são os métodos invocados pelo OSCListener quando chega uma mensagem. Estes comandos podem ser agrupados em três grupos: Send, Info, Broadcast

4.2.2.1 Comandos Send

Os comandos Send são comandos usados para enviar um ficheiro para um ou mais dispositivos. Dentro deste grupo estão os comandos SendPath e SendPathWithMime, devido à sua grande semelhaça. Estes dois comandos usam as funcionalidades do pacote Bluetooth para enviar ficheiros para dispositivos móveis. Cada um destes comandos pode enviar um ficheiro para um dispositivo ou lista de dispositivos. O método de envio para vários dispositivos não é mais do que replicar o envio de um ficheiro para um dispositivo de cada vez. A diferença entre estes

comandos é que o SendPath usa um *mimetype* pré-definido, enquanto que o comando SendPathWithMime usa o *mimetype* definido na mensagem OSC.

Outros dois comandos que se podem agrupar são os comandos SendPathToGroup e sendPathWithMimeToGroup. Estes dois comandos enviam um ficheiro para um grupo de dispositivos, sendo o grupo definido na mensagem OSC. Um grupo é definido por três parâmetros: *major class device*, *minor class device*, e marca. O ficheiro só é enviado para dispositivos que pertençam ao grupo definido por estes três parâmetros. O funcionamento deste comando depende da classe GroupGetter, que é responsável por identificar os dispositivos que pertencem ao grupo. Após ter uma lista de dispositivos, são usados os comandos SendPath e SendPathWithMime para fazer a entrega dos ficheiros.

4.2.2.2 Comandos Info

Existem dois comandos de informação o GetGroup e o GetFileList.

GetGroup – Este comando procura todos os dispositivos ao alcance da aplicação DiABlu MailMan que pertençam ao grupo definido na mensagem OSC. A lista destes dispositivos é posteriormente enviada de volta para a aplicação que enviou a mensagem OSC.

GetFileList – Este comando retorna duas listas de ficheiros: a lista de ficheiros enviados e a lista de ficheiros recebidos por um determinado dispositivo definido na própria mensagem. É de salientar que não é necessário que o dispositivo esteja ao alcance da aplicação para que esta informação seja obtida; estas listas de ficheiros são mantidas dentro da própria aplicação, e são arquivadas para que possam ser usadas em sessões posteriores.

4.2.2.3 BroadCast

Existem dois comandos deste tipo, o Broadcast, e o BroadcastWithMime. Estes comandos são bastante semelhantes aos comandos SentPathToGroup e SendPathToGroupWithMime respectivamente. A diferença entre os dois grupos de comandos é que os dois iniciados com a palavra Broadcast são comandos temporais, ou seja, enviam ficheiros para determinado grupo de dispositivos que entrem no alcance da aplicação DiABlu MailMan num intervalo de tempo definido. Devido à possibilidade do uso da *wildcard* '*' na definição dos grupos, estes comandos tem a possibilidade de enviar ficheiros para um grupo que é constituído por qualquer dispositivo.

Os comandos de Broadcast têm dois parâmetros relacionados com o tempo. O primeiro parâmetro é a duração do intervalo do tempo em que devem ser enviados ficheiros para o grupo. O segundo parâmetro está relacionado com a espera entre pesquisas de dispositivos. O processo de pesquisa de dispositivos é um processo

bastante moroso e além disso, enquanto este processo ocorre, é impossível iniciar o envio e recepção de ficheiros, devido a limitações da tecnologia Bluetooth. Como tal, a espera entre pesquisas deve ser grande o suficiente para que o processo de pesquisa de dispositivos não represente uma parcela muito grande do tempo total.

4.3 Interface gráfica

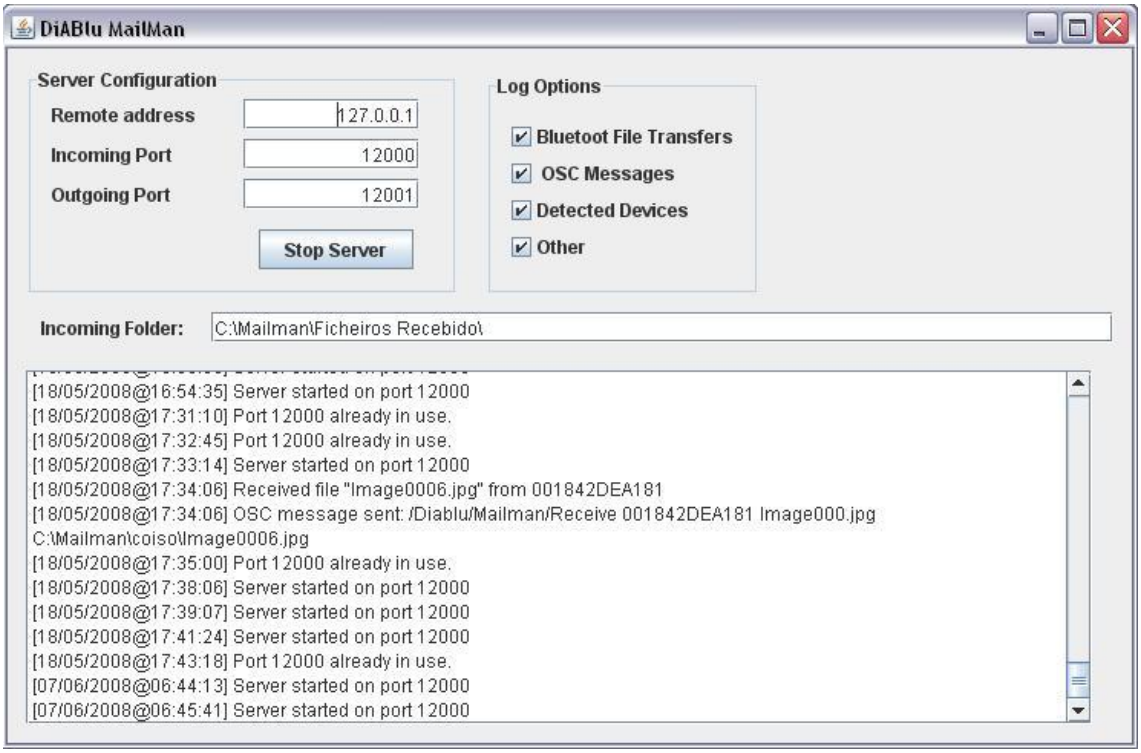

Figura 4.4 - Interface gráfica do DiABlu MailMan

A interface gráfica do DiABlu MailMan é bastante simples e intuitiva. É composta por um pequeno grupo de configuração do servidor OSC, onde se pode definir o endereço IP e porta da máquina ao qual a aplicação DiABlu MailMan se vai ligar, assim como a porta que esta aplicação disponibiliza para ligações OSC.

Está também presente nesta interface uma caixa de texto onde se pode escolher o directório onde vão ser guardados os ficheiros que chegam por Bluetooth.

Existe também a possibilidade de configurar os *logs* que são mostrados na grande caixa de texto, em baixo. Estas opções permitem mostrar ou esconder os diferentes tipos de *logs* para facilitar a visualização.

4.4 Classes utilitárias

Das classes utilitárias existentes na aplicação DiABlu MailMan apenas duas são de especial relevo: o GroupGetter e o Logger.

4.4.1.1 GroupGetter

O GroupGetter é a classe usada nos comandos SendToGroup e BroadCast. Esta classe tem os métodos necessários para escolher, de todos os dispositivos existentes ao alcance da aplicação DiABlu MailMan, aqueles que pertencem a um grupo específico.

O funcionamento desta classe está dependente da classe Discovery do pacote Bluetooth. É essa classe que irá fornecer uma lista de todos os dispositivos ao alcance da aplicação DiABlu MailMan. Assim que a lista é obtida, ela é passada por três filtros, um por cada parâmetro que define um grupo. No final, obtêm-se a lista dos dispositivos que pertencem ao grupo pretendido. Todos os dispositivos que não pertenciam ao grupo foram eliminados pelos filtros.

Com esta lista de dispositivos é possível usar os comandos SendPath para distribuir os ficheiros por todos os dispositivos presentes na lista.

4.4.1.2 Logger

Esta classe foi criada por motivos de personalização em alternativa ao Logger existente em Java o que apresenta duas grandes vantagens:

- Permite classificar os *logs*. Desta forma podemos saber a que é que os *logs* são relativos. Isto permite ainda filtrar os *logs* que são visíveis na interface gráfica.
- Trata os *logs* quando são criados. No caso da aplicação DiABlu MailMan, os *logs* são apresentados na interface gráfica, são apresentados na consola, e são ainda guardados num ficheiro, para que possam ser analisados mais tarde.

4.5 Estruturas de dados

4.5.1 MailManDevice e MailManKnownDevices

Estas duas classes foram criadas com o objectivo de manter uma lista de dispositivos, que de uma maneira ou de outra, interagiram com a aplicação DiABlu MailMan.

A estrutura de dados MailManDevice guarda toda a informação disponível sobre um dispositivo. Assim, além dos dados característicos do dispositivos, como por exemplo o endereço Bluetooth, o *major device classe,* o *minor device class,* etc. é também guardada uma lista de ficheiros enviados por este dispositivo, assim como uma lista de ficheiros recebidos pelo mesmo.

A estrutura MailManKnownDevices é a estrutura que guarda as instâncias da classe MailManDeviceClass. De modo a facilitar a pesquisa dos dispositivos, esta estrutura é uma HashTable, pois o acesso a uma posição especifica é directo, o que não acontece por exemplo nos Vectores.

4.6 Resumo do capítulo

Neste capítulo foram apresentados os detalhes da implementação da aplicação DiABlu MailMan. Começou por se expor os dois pacotes ligados à comunicação da aplicação com o exterior, o pacote Bluetooth e o pacote OSC. No pacote Bluetooth foi possível ver como se processa o envio e a recepção de um ficheiro. No pacote OSC explicou-se como funciona o envio e recepção de mensagens, e quais as suas consequências. Seguidamente foi apresentada a interface gráfica da aplicação. Por último foram expostas as classes utilitárias e as estruturas de dados mais relevantes no desenvolvimento da aplicação.

Capítulo 5

Avaliação, demonstrações e aplicações

5.1 Avaliação

Chegado o final do projecto, os resultados obtidos são bastante satisfatórios. A aplicação funciona nos dois sistemas operativos alvo e quase todos os requisitos foram cumpridos. O único requisito que ficou por implementar foi a presença de ferramentas de teste de comunicação por Bluetooth, inseridas no painel de configuração da interface gráfica.

A aplicação tem-se mostrado robusta e tolerante a falhas, embora um teste de esforço não tenha sido ainda efectuado. No entanto muitos testes foram feitos, com um sentido crítico, na tentativa de atacar a aplicação onde estaria mais fragilizada. Desses testes resultaram algumas correcções, mas na aplicação mostrou-se capaz de resolver todos os problemas atirados para o seu caminho.

Estes testes dão algumas certezas de que esta aplicação é fiável, mas mais uma vez não é possível tirar conclusões concretas enquanto não for realizado um teste de esforço.

5.2 Demonstrações e aplicações

No decorrer do desenvolvimento da aplicação foram desenvolvidas algumas demonstrações para testar o funcionamento da aplicação e para comprovar a utilidade da mesma.

5.2.1 Receive File

Esta demonstração é um exemplo do que pode ser feito usando a aplicação DiABlu MailMan. Este exemplo foi desenvolvido em Processing. O código deste exemplo encontra-se em anexo.

Figura 5.1 - Ecrã inicial da demonstração

Quando a demonstração é inicializada, pede que lhe seja enviada uma imagem, como se mostra na figura 5.1. Quando esta demonstração recebe uma imagem o ecrã apresentado muda para o da figura

Figura 5.2 - Ecrã apresentado pela demonstração após a recepção de um ficheiro

5.2.2 Snap&Share

O Snap&Share é uma instalação de arte digital desenvolvida usando a aplicação DiABlu MailMan. Esta instalação recebe fotografias enviadas de um dispositivo móvel, por Bluetooth, e apresenta essas mesmas fotografias num *display*. As fotografias recebidas são mostradas de forma aleatória numa grelha de tamanho variável entre 1x1 e 4x4.

Esta instalação foi desenvolvida por Jorge Cardoso, Pedro Baptista e Pedro Santos. Esteve em exibição no Mosteiro de Leça do Balio durante uma festa organizada pela Progressive Sounds of Porto (PSP).

Embora a instalação tivesse desempenhado o seu papel, teve pouca aderência por parte do público. Esta fraca adesão deveu-se ao facto de as imagens serem projectadas em tule demasiado fino, numa sala com demasiada luminosidade, o que tornava bastante difícil a correcta visualização das fotografias.

Apesar dos resultados pouco satisfatórios, o projecto continua, assim como continua a busca por um local melhor para a instalação.

5.2.3 História Cooperativa

A História Cooperativa é uma instalação em que o público conta uma história. O público vai contando a história através de ficheiros de texto enviados por Bluetooth. Como é uma história que conta com a participação de diversas pessoas, o seu rumo vai ser alterado diversas vezes durante o seu curso.

Quando a história chegar ao fim, uma cópia da história será enviada para todos os dispositivos Bluetooth presentes, de modo a que o público fique com uma recordação daquilo que ajudou a construir

Esta instalação ainda não chegou à fase final do seu desenvolvimento, mas estará pronta em breve.

Capítulo 6

Conclusões e trabalho futuro

6.1 Resumo

Este documento tem como objectivo principal relatar o desenvolvimento da aplicação DiABlu MailMan incluindo as alternativas e justificando as principais opções. Inicialmente foi exposto o estudo efectuado sobre o estado da arte. Verificou-se que, das soluções encontradas, nenhuma tem a capacidade de efectuar o papel da aplicação DiABlu MailMan. Todas são dedicadas ao marketing de proximidade, quando o pretendido é uma aplicação orientada às artes digitas, interactiva por definição.

O passo seguinte foi fazer uma análise tecnológica. Nesta análise foram estudadas as melhores soluções tecnológicas para o projecto, como é o caso dos protocolos de comunicação Bluetooth e OSC. Foram também explicadas as razões pelas quais a linguagem de programação Java foi escolhida. Foi neste capitulo ainda que se comparou diferentes bibliotecas e se escolheu a melhor.

No capítulo seguinte deste documento foi exposta toda a análise de requisitos e arquitectura do sistema. Esta secção demonstra o trabalho de análise feito previamente à fase de implementação.

O capítulo quatro efectua uma descrição de pormenores de implementação. Neste capítulo é explicado o funcionamento da aplicação, a mais baixo nível, apresentando-se as principais classes, e estruturas de dados.

Finalmente é efectuada uma a avaliação do trabalho implementado, e a apresentação das demonstrações e das aplicações que foram desenvolvidas usando as funcionalidades Bluetooth da aplicação DiABlu MailMan.

6.2 Conclusões

Após concluir este projecto, verificou-se que o protocolo Bluetooth é uma tecnologia muito avançada, embora esteja ainda limitada em alguns aspectos, como é o caso do consumo de energia. É um protocolo bastante difícil de usar, devido a falhas inexplicáveis e inconsistências entre implementações. No entanto, havendo dedicação suficiente esses obstáculos podem ser ultrapassados, permitindo assim o acesso a uma tecnologia que começa agora a aparecer, cada vez mais num ambiente para o qual não foi desenvolvida.

O projecto DiABlu é a prova disto, pois pegou nesta tecnologia e com ela produziu ferramentas que podem ser usadas no meio das artes digitais.

O protocolo de comunicação OSC surpreende pela positiva, pois é um protocolo bastante simples, mas eficaz. Devido à popularidade deste protocolo, está presente na maioria das aplicações usadas no âmbito das artes digitais, o que o torna na escolha ideal para o projecto.

Uma das funcionalidades mais importantes deste projecto prende-se com a gestão de utilizadores. É possível, a qualquer momento, determinar quantos e quais os utilizadores presentes, assim como toda a informação sobre a sua interacção, que será guardada, de modo a que possa ser usada em sessões posteriores. Deste modo é possível adaptar a instalação de arte digital ao ambiente no qual está inserida, podendo alterar o seu comportamento de acordo com o número de utilizadores presentes, ou até com o fluxo de interacções.

O facto de se ter definido um dialecto para as mensagens OSC, permite que haja uma fácil integração com as aplicações usadas na criação de arte digital, tornando simples e intuitivo o uso desta ferramenta.

Apesar de ainda não ter sido muito usada, já se começa a perceber a importância que a aplicação DiABlu MailMan terá nas artes digitais. Introduz uma nova forma de interacção que, bem utilizada, nas mãos de alguns artistas digitais poderá produzir resultados memoráveis.

6.3 Trabalho futuro

Embora a aplicação esteja pronta não foi ainda possível realizar testes de esforço da aplicação. Este teste consiste em manter a aplicação a funcionar durante um período de tempo alargado, com um grande número de transferências efectuadas; Este teste servir para testar o comportamento da aplicação, assim como a compatibilidade com dispositivos de diferentes marcas.

É também desejável ter concluído a instalação História Cooperativa. É uma instalação bastante interessante, que se tiver aceitação do público pode servir como rampa de lançamento da aplicação DiABlu MailMan.

Um melhoramento técnico a introduzir é a correcção da biblioteca NetUtil de modo a que fosse possível utilizar o protocolo TCP para enviar mensagens de tamanho superior a 8kb.

Tem também interesse desenvolver mais demonstrações e instalações de arte digital, que contém com a aplicação DiABlu MailMan, de forma a dar a conhecer aos artista digitas.

Referências e Bibliografia

[1] Jorge Cardoso. DiABlu Project. http://diablu.jorgecardoso.eu/ (acedido em 07/07/08).

[2] Open Sound Control. http://opensoundcontrol.org/ (acedido em 07/07/08).

[3] Digital Art, Wikipedia. http://en.wikipedia.org/wiki/Digital_art (acedido em 07/07/08).

[4] M. Puckette. *Pure Data: another integrated computer music environment*. Em *Proc. The Second Intercollege Computer music concert, páginas 37-41, 1996.*

[5] Ben Fry and Casey Reas. Processing. http://processing.org (acedido em 07/07/08).

[6] InfoMus Lab. EyesWeb. http://musart.dist.unige.it/EywMain.html (acedido em 25/03/08).

[7] Cycling74. Max/MSP. http://www.cycling74.com (acedido em 25/03/08).

[8] Adobe. Flash. http://www.adobe.com (acedido em 25/03/08).

[9] Bluetooth.com | Basics http://www.bluetooth.com/Bluetooth/Technology/Basics.htm (acedido em 07/07/08)

[10] Jose Soares. *Bluetooth Factory na Shortcut. Relatório de estágio da LEIC na FEUP. 2007*

[11] Shortcut – A empresa. http://www.shortcut.pt/empresa.php (acedido em 07/07/08).

[12] blue cell network. http://www.bluecellnetworks.com/ (acedido em 07/07/08)

[13] BeamZone. http://www.bluecellnetworks.com/en/beamzone.html (acedido em 07/07/08).

[14] Qwikker. http://www.qwikker.com/ (acedido em 07/07/08).

[15] FuturLink: Inspiring Mobile Media. http://www.futurlink.com/en/company.php (acedido em 07/07/08).

[16] Wilico. http://futurlink.com/en/products.php (acedido em 07/07/08

[17] BlueSender. http://www.bluesender.com/ (acedido em 07/07/08)

[18] Medieval Software – Medieveval Bluetooth File Transfer. http://www.medieval.it/content/view/17/56/ (acedido em 07/07/08)

[19] Java Technology. http://www.sun.com/java/ (

[20] Sun Microsystems. http://www.sun.com/ (acedido em 07/07/08)

[21] Java Virtual Machine – Wikipédia. http://en.wikipedia.org/wiki/Java_Virtual_Machine (acedido em 07/07/08)

[22] Bluetooth.com. http://www.bluetooth.com/bluetooth/ (acedido em 07/07/08)

[23] Bluetooth.com | OBEX. http://www.bluetooth.com/Bluetooth/Technology/Works/OBEX.htm (acedido em 07/07/08)

[24] JSR-82: Java Bluetooth. http://www.bluetooth.com/Bluetooth/Technology/Works/OBEX.htm (acedido em 07/07/08)

[25] Bluecove. http://code.google.com/p/bluecove/ (acedido em 07/07/08)

[26] avetanaBluetooth. http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml (acedido em 07/07/08)

[27] NetUtil. http://www.sciss.de/netutil/ (acedido em 07/07/08)

[28] JavaOSC. http://www.illposed.com/software/javaosc.html (acedido em 07/07/08)

Anexo A: Código da demonstação

```
import netP5.*;
import oscP5.*;

OscP5 oscServer;

PFont frontPage;
PFont title;
PFont fileInfo;
File file;


boolean hasFile = false;

void setup()
{
  frameRate(25);
  size(640,480);
  background(0);

  frontPage = loadFont("HarlowSolid-48.vlw");
  title = loadFont("HarlowSolid-32.vlw");
  fileInfo = loadFont("Arial-BoldMT-13.vlw");

  oscServer = new OscP5(this, 12001, OscP5.UDP);
}

void draw()
{
  background(0);
  if(hasFile == true)
  {
    textFont(frontPage);
    textAlign(CENTER);
    fill(128, 128, 128);


    text("Recieved File", 320, 60);

    textFont(fileInfo);

    textAlign(RIGHT);

    text(file.path, 600, 120);
    text(file.filename, 600, 140);
    text(file.directory, 600, 160);
    text(file.ext, 600, 180);

    textAlign(LEFT);

    text("Path:", 40, 120);
    text("Filename:", 40, 140);
    text("Directory:", 40, 160);
    text("Extension:", 40, 180);

    textAlign(CENTER);
    text("Sent by device " + file.device + ".", 320, 460);
    if(file.readyToPrint)
    {

      tint(256, 128, 0);
      image(file.img, width/2 - file.w, 270 - file.h/2,file.w, file.h);
      tint(256, 0, 0);
      image(file.img, width/2 - file.w, 270 + file.h/2,file.w, file.h);
```

```
      tint(0, 256, 0);
      image(file.img, width/2, 270 - file.h/2, file.w, file.h);
      tint(0, 0, 256);
      image(file.img, width/2, 270 + file.h/2, file.w, file.h);

    }
  }
  else
  {
    fill(128, 128, 128);
    textFont(frontPage);
    textAlign(CENTER);
    text("Send a picture, please...", width/2, height/2);

  }
}

void oscEvent(OscMessage theOscMessage)
{
  file = new File(theOscMessage);
  hasFile = true;
}

class File{

  String path;
  String filename;
  String directory;
  String ext;
  String device;
  PImage img;
  PImage bwImg;
  PImage redImg;
  PImage blueImg;
  PImage greenImg;
  boolean readyToPrint = false;
  float w;
  float h;
  float k;
  boolean isImage = false;


  public File(OscMessage oscMsg)
  {
    readyToPrint = false;

    this.device = (String) oscMsg.arguments()[0];
    this.filename = (String) oscMsg.arguments()[1];
    this.path = (String) oscMsg.arguments()[2];

    process();

  }

  private void process()
  {

    String tempdir = path;
    directory = "";

    while(tempdir.indexOf('\\') != -1)
    {
      directory += tempdir.substring(0, tempdir.indexOf('\\')+1);
      tempdir = tempdir.substring(tempdir.indexOf('\\') +1);
    }

    ext = filename;
```

```
    while(ext.indexOf('.') != -1)
    {
      ext = ext.substring(ext.indexOf('.') +1);
    }
    ext = ext.toLowerCase();
    if(ext.equals("gif") == true)
    {
      isImage = true;
    }
    if(ext.equals("jpg") == true)
    {
      isImage = true;
    }
    if(ext.equals("tga") == true)
    {
      isImage = true;
    }
    if(ext.equals("png") == true)
    {
      isImage = true;
    }



    if(isImage)
    {

      img = loadImage(path);
      k = 100.0 / img.height;
      h = 100;
      w = img.width * k;
      if(w > 300)
      {
        k = 300.0 / w;
        w = 300;
        h = h*k;
      }
      readyToPrint = true;
    }

  }

}
```