Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado em Engenharia Informática
2º Semestre, 2009/2010

Framework Para Aplicações Colaborativas Multi-Plataforma

Hugo Filipe Madeira Fernandes, 28062

Orientadores
Prof. Doutor Vítor M. A. Duarte – FCT/UNL
Eng. Paulo Chainho – PT Inovação

28 de Julho de 2010

N.º de Aluno: 28062
Nome: Hugo Filipe Madeira Fernandes

Título da Dissertação:
Framework Para Aplicações Colaborativas Multi-Plataforma

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

Palavras-Chave:

- PUC – Plataforma Unificada de Colaboração
- Framework Colaborativa
- Desktop/Web Widgets
- OpenMeetings
- Aplicações de Internet Rica

Keywords:

- PUC – Plataforma Unificada de Colaboração
- Collaborative Framework
- Desktop/Web Widgets
- OpenMeetings
- Rich Internet Application

Resumo

Com a massificação de dispositivos móveis "inteligentes", também conhecidos por *smartphones*, e o crescente aumento do número destes ligados à internet, surge o desafio de desenvolver aplicações colaborativas viradas para este tipo de dispositivos. Contudo, para isso, é necessário ter um conhecimento das suas limitações de modo a poderem ser feitas as melhores escolhas.

Neste sentido, nesta dissertação estudou-se a viabilidade da implementação de funcionalidades colaborativas como videoconferência, *whiteboard*, *remote desktop*, *desktop sharing* e visualização de diapositivos, em dispositivos móveis.

De modo a alcançar este objectivo foi necessário estudar quais as tecnologias de desenvolvimento existentes, os diversos protocolos e os requisitos necessários para fazer face aos desafios levantados. Foi, também um objectivo bastante importante que aquando da tomada de decisão da escolha da tecnologia a usar esta tenha a maior transversalidade e ubiquidade possível. Tem que existir um ponto de convergência entre este último requisito e os outros necessários ao desenvolvimento de aplicações que suportem as supra citadas funcionalidades colaborativas.

Neste trabalho procedeu-se ao desenvolvimento de um conjunto de widgets que permitem o fácil desenvolvimento de aplicações com este tipo de funcionalidades.

Abstract

With the increasing number of mobile devices, specially smartphones, and the growing number of those connected to the Internet, comes the challenge of developing collaborative applications in order to take full advantages of this devices. However, you must be aware of their limitations so you can make the best choices.

Therefore, this master thesis studied the feasibility of establishing collaboration features such as video conferencing, whiteboard, remote desktop, desktop sharing and viewing of slides, on desktop and mobile devices.

To achieve this goal it was necessary to study the development technologies, the various protocols and requirements necessary to meet the challenges. It was also a very important goal to chose the technology that supports the largest cross-section and ubiquity possible. There must be a point of convergence between the latter and the other requirement necessary for developing applications that support the above mentioned collaboration features.

In this work we developed a set of widgets that enable easy development of applications with this type of functionality.

Índice

1. Introdução

1.1. Motivação

Estamos na Era da Globalização, onde se pede a participação de todos os povos e, acima de tudo, das pessoas. Neste novo paradigma, onde vivemos, é-nos exigido encurtar distâncias. Caminhamos, cada vez mais, para um mundo onde as distâncias tendem a ser atenuadas pela evolução tecnológica. É pois, neste sentido, que tende a caminhar muita da inovação na área das comunicações. Com o intuito de aproximar as pessoas, muito do que de novo existe tem como objectivo estimular a ligação e colaboração entre todos.

O interesse pela área da colaboração data já de alguns anos. Em 1984, Irene Greif e Paul M. Cashman introduziram o conceito de *computer supported cooperative work* (CSCW) num *workshop* para pessoas que pretendiam usar tecnologia como meio de suporte no trabalho[1]. O conceito é algo vasto mas em 1991, Wilson[2] definiu-o da seguinte maneira: "*CSCW a generic term, which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques.*". Desde então, acompanhado pela melhoria das infra-estruturas que podem dar suporte à área da colaboração à distância, tem-se vindo a percorrer um caminho que possa ir ao encontro das necessidades das pessoas.

Actualmente, o interesse na área da colaboração e em serviços que possam aproximar distâncias continua presente, como atesta o lançamento de serviços Google que permitem superar esse desafio. São exemplos disso, o Google Voice And Video Chat

e o Google Docs. O primeiro oferece uma solução de videoconferência e o segundo ferramentas para edição colaborativa de documentos, whiteboard, entre outras.

Números oficiais indicam que no final do ano de 2008 existiam mais de 4 mil milhões de telemóveis no mundo[3]. A par deste crescimento surge o aumento do seu poder computacional e da velocidade e fiabilidade da internet móvel. Muitos dos serviços hoje oferecidos pressupõe uma ligação constante a esta rede.

Assim, surge a motivação para analisar, explorar e testar os conceitos que permitam trazer este tipo de ferramentas colaborativas para os dispositivos móveis, sem descurar os restantes. É, pois, necessário estudar a viabilidade da criação de mecanismos que permitam a maior transversalidade possível e plataformas que facilitem o desenvolvimento desta classe de aplicações.

1.2. Descrição do Problema e Objectivos

1.2.1. Contextualização

Os ambientes colaborativos, neste contexto, são "espaços" virtuais que tornam possível pessoas dispersas geograficamente confluírem num mesmo local. Este conceito pode ser aplicado em diferentes contextos, seja em ambientes profissionais ou mais lúdicos. Numa vertente mais profissional, por exemplo, é possível, através de um ambiente colaborativo organizar uma reunião e assim fornecer a capacidade de partilhar informação, trocar pontos de vista e, por fim, chegar a um entendimento sobre uma possível tomada de decisão. De um ponto de vista mais lúdico, é possível criar um espaço virtual onde um conjunto de amigos se reuna e fale para combinar as próximas férias.

De modo a podermos responder aos desejos das pessoas no que toca à comunicação e à criação de ambientes colaborativos, é, cada vez mais, necessário disponibilizar ferramentas que possam contribuir para o desenvolvimento de aplicações que preencham essa necessidade. No exemplo acima referido, sobre a reunião, talvez dê jeito existir um quadro virtual onde possam ser feitos esquemas gráficos ou a possibilidade do orador poder partilhar com os restantes participantes os seus diapositivos. Seria ainda interessante que todos se pudessem ver e ouvir, como se de uma reunião física se tratasse. Assim, entre as várias ferramentas existentes, podemos considerar alguns tipos base de aplicações como a videoconferência, *remote desktop* e *desktop sharing*, visualização de diapositivos, *whiteboard* e a possibilidade de partilha de ficheiros, a seguir descritas:

Videoconferência consiste na transmissão de áudio e vídeo, em tempo real, entre pessoas que estão fisicamente distantes, mas dando a sensação que se encontram no mesmo local. Usualmente, este método permite comunicações pessoa-a-pessoa ou em grupo. A tecnologia de base usada neste serviço é a transmissão do áudio e video, em tempo real. Para tal é efectuada a compressão do sinal e subdividido em pacotes que são enviados pela rede (normalmente sobre IP ou ISDN). Na recepção, o processo aplicado é o inverso, havendo a descompressão dos dados e reconstrução do sinal. Um dos produtos

mais conhecidos do mercado é o TelePresence da Cisco. Existem ainda alguns estudos e implementações académicas como a realizada pela Universidade de Washington, chamada *Odegaard Videoconference Studio*[4].

Remote Desktop permite que um utilizador remoto se ligue a outro computador e, de algum modo, o possa ver e controlar. Consideremos o caso em que uma pessoa solicita a intervenção de um técnico pelo serviço de *helpdesk*. Através de *remote desktop* é possível que o técnico aceda ao computador da pessoa e que, remotamente, se possa realizar a intervenção fazendo uso, inclusive, da interface gráfica. Quando tal acontece, o computador que acede recebe uma imagem do visor do computador acedido que é actualizada de tempos a tempos ou quando se dá alguma mudança neste último. Para além disto, o computador que acede, envia os seus eventos do rato e teclado que são, então, recebidos pelo computador controlado e lá aplicados. Uma conhecida aplicação que aplica esta capacidade é o VNC (Virtual Network Computing). Embora já existam bastantes aplicações que implementam esta funcionalidade entre computadores (*desktop*), são poucas[5] as que permitem aceder a um computador (*desktop*) através de um dispositivo móvel. Existem ainda alguns desafios a ser ultrapassados como os referidos por Khaled Khankan e Robert Steele[6] e que se prendem, entre outras, com questões de segurança e interacção através de um dispositivo móvel.

Desktop Sharing é a possibilidade de partilha do ambiente de trabalho com vários utilizadores. Através desta funcionalidade é possível que um grupo de pessoas visualize o que um outro utilizador está a fazer no seu computador. Esta capacidade torna-se útil quando, por exemplo, queremos demonstrar algo a alguém ou a um grupo de utilizadores. É assim uma maneira de ultrapassar a distância física. Esta funcionalidade pode ser encarada como *remote desktop* sem a capacidade de intervenção no computador remoto.

A visualização colaborativa de diapositivos permite que utilizadores dispersos fisicamente possam vê-los, cada um no seu terminal. Supondo que o utilizador A convida os utilizadores B e C para partilhar os seus diapositivos, é possível que B e C estejam a ver, de forma sincronizada, o mesmo diapositivo mostrado por A. Isto é possível, enviando A, por exemplo, o seu *powerpoint* para o servidor que se encarregará de converter em imagens, distribuir e sincronizar com os restantes participantes.

Whiteboard é um espaço comum em que múltiplos utilizadores podem trabalhar em simultâneo[7]. Funciona como um quadro virtual em que os diversos intervenientes podem escrever e todos os outros verem. Torna-se uma ferramenta útil aquando, por exemplo, de uma reunião para a elaboração de um esboço sobre a arquitectura de um projecto. Uma solução existente no mercado, mas apenas para Windows, é a da Nefsis[8].

Por último, a possibilidade de partilha de ficheiros sugere a funcionalidade de enviar ficheiros (*upload*), de modo a ficarem disponíveis para posterior partilha e descarga (*download*) ou, porventura, troca directa entre utilizadores (*peer-to-peer*).

Apesar da já existência destas tecnologias, surge o problema da sua fácil integração em aplicações concretas e possivelmente acedidas por dispositivos móveis. Hoje em dia, no mercado, existem inúmeras marcas e modelos de telemóveis, portanto, o desafio que se impõe é o da criação de uma estrutura que permita a maior transversalidade possível. No entanto, este objectivo acarreta ainda outro problemas. Os dispositivos móveis, ao contrário dos computadores, têm baixa resolução e normalmente não possuem periféricos como o rato ou teclado o que causa algumas limitações que têm que ser ultrapassadas, usualmente, através de versões da aplicação específicas. A juntar a isto existe ainda o problema da ligação à internet. Neste tipo de dispositivos é comum haver muitas falhas com banda e latência limitadas que precisam ser tidas em conta. É necessário reduzir a comunicação ao indispensável, adaptar parâmetros como por exemplo a qualidade de um vídeo e adoptar métodos de persistência de modo ao utilizador não perder dados caso perca a ligação.

Um outro problema existente prende-se com a escolha da tecnologia de desenvolvimento a usar. É necessário adoptar uma que permita fazer face aos problemas acima apresentados. Um requisito importantíssimo na escolha da mesma é a sua transversalidade; convém que o resultado produzido seja o mais compatível possível com todos os dispositivos onde a aplicação será usada.

Associado à problemática da escolha da tecnologia de suporte a usar é importante analisar os protocolos que com elas podem trabalhar. É necessário perceber, para além da compatibilidade entre o protocolo e a tecnologia a adoptar, se este responde aos requisitos necessários para o desenvolvimento das funcionalidades multimédia acima mencionadas.

1.2.2. Objectivos

Enquadrado num projecto maior (PUC, explicado abaixo) cujo objectivo é a criação de uma plataforma única de colaboração, pretende-se estudar e, posteriormente, desenvolver um mecanismo que permita ultrapassar as dificuldades de implementação, do tipo de características acima descritas, na maior variedade possível de dispositivos móveis e não só. É, assim, o nosso propósito estudar uma solução que torne possível o desenvolvimento de novas aplicações independentes do sistema operativo móvel a aplicar. Isto permitirá que quem desenvolva as referidas aplicações se possa abstrair dos aspectos mais técnicos de cada dispositivo e das funcionalidades colaborativas básicas, permitindo uma maior reutilização do código, com consequente melhoria na qualidade e produtividade do desenvolvimento.

Em suma, o objectivo principal assenta no estudo e desenvolvimento de componentes gráficas multimédia reutilizáveis, como *widgets*, para o desenvolvimento de novas aplicações de colaboração nos terminais, em particular em dispositivos móveis. Esses componentes devem permitir a partilha de recursos multimédia incluindo as acima mencionadas (videoconferência, r*emote resktop* e *desktop sharing*, visualização colaborativa de diapositivos, *whiteboard* e partilha de ficheiros). Não é também de descurar as intervenções que tenham que ser feitas na plataforma (PUC) de modo a tornar possível o desenvolvimento do que é proposto.

No final desta dissertação, espera-se ter conseguido desenvolver os componentes já mencionados e que estes possam funcionar em diferentes dispositivos heterogéneos, oferecendo a capacidade de aplicações cliente facilmente utilizarem as funcionalidades colaborativas acima referidas.

No mercado já existe uma solução disponibilizada pela Adobe, o ConnectNow[9], que fornece todas as funcionalidades (videoconferência, *remote desktop*, etc.) acima referidas. Poder-se-ia pensar que tal solução serve os interesses do projecto desenvolvido, contudo existem vários inconvenientes. O principal, é o facto de serviço da Adobe não ser uma solução modular. Como foi referido, no âmbito desta dissertação

pretendem-se criar *widgets* que possam ser usados por outras aplicações. Com o ConnectNow, não é, de todo, isso que acontece. Um outro ponto negativo prende-se com o facto de a solução da Adobe ser de código fechado, tornando assim impossível a sua utilização enquadrada noutros projectos ou aplicações. Deste modo, a existência deste produto não inviabiliza o trabalho realizado.

1.3. Estrutura do Relatório

O restante documento está dividido em mais 6 capítulos que de seguida são enunciados.

O capítulo 2, PUC – Plataforma Unificada de Colaboração, abordará a plataforma sobre a qual o projecto foi desenvolvido e está dividido em duas subsecções: Descrição da Plataforma e Trabalho Relacionado. A primeiro, como o nome sugere, descreve as principais características da plataforma bem como a sua arquitectura de alto nível. A segunda, refere o trabalho já existente à cerca do PUC.

O capítulo 3, Tecnologias Para o Desenvolvimento, servirá para analisar diferentes tecnologias relacionadas com o trabalho que se pretendeu desenvolver, nomeadamente ferramentas e protocolos que possibilitem alcançar o objectivo proposto.

No capítulo 4, Desenho da Solução, será apresentado o desenho da solução a implementar.

O capítulo 5, Implementação, corresponderá à secção onde serão discutidos os detalhes de implementação e será dividida em duas subsecções: Cliente e Servidor. Nelas falar-se-ão do trabalho desenvolvido na componente de cliente e servidor, respectivamente.

O capítulo 6, Testes e Avaliação, será o que irá conter os testes e avaliação realizados. Concretamente, conterá os testes de funcionalidade, portabilidade e desempenho.

Por último, no capítulo 7, Conclusões, encontrar-se-ão as conclusões do trabalho realizado, bem como o trabalho futuro.

2. PUC – Plataforma Unificada de Colaboração

2.1. Descrição da Plataforma

A realização deste trabalho insere-se num projecto maior, actualmente em desenvolvimento na PT Inovação, que dá pelo nome de PUC – Plataforma Unificada de Colaboração. De seguida será apresentado, sumariamente, o projecto.

PUC é uma *framework* modular que pretende oferecer uma solução unificada para soluções dos mais diversos tipos de colaboração. A partir desta plataforma pretende-se ser possível gerir toda a lógica associada às funcionalidades de colaboração, facilitando a tarefa a quem desenvolve aplicações deste tipo. Para isso faz uso de diferentes recursos e fornece interfaces para quem pretende desenvolver aplicações ou adicionar módulos que ofereçam novas funcionalidades. Tem como grande objectivo disponibilizar ferramentas de gestão de participantes, grupos e sessões, bem como recursos comummente associados a aplicações colaborativas, tais como troca de mensagens, edição de texto, video-conferência, *whiteboard*, entre outros.

Eis a arquitectura resumida do PUC:

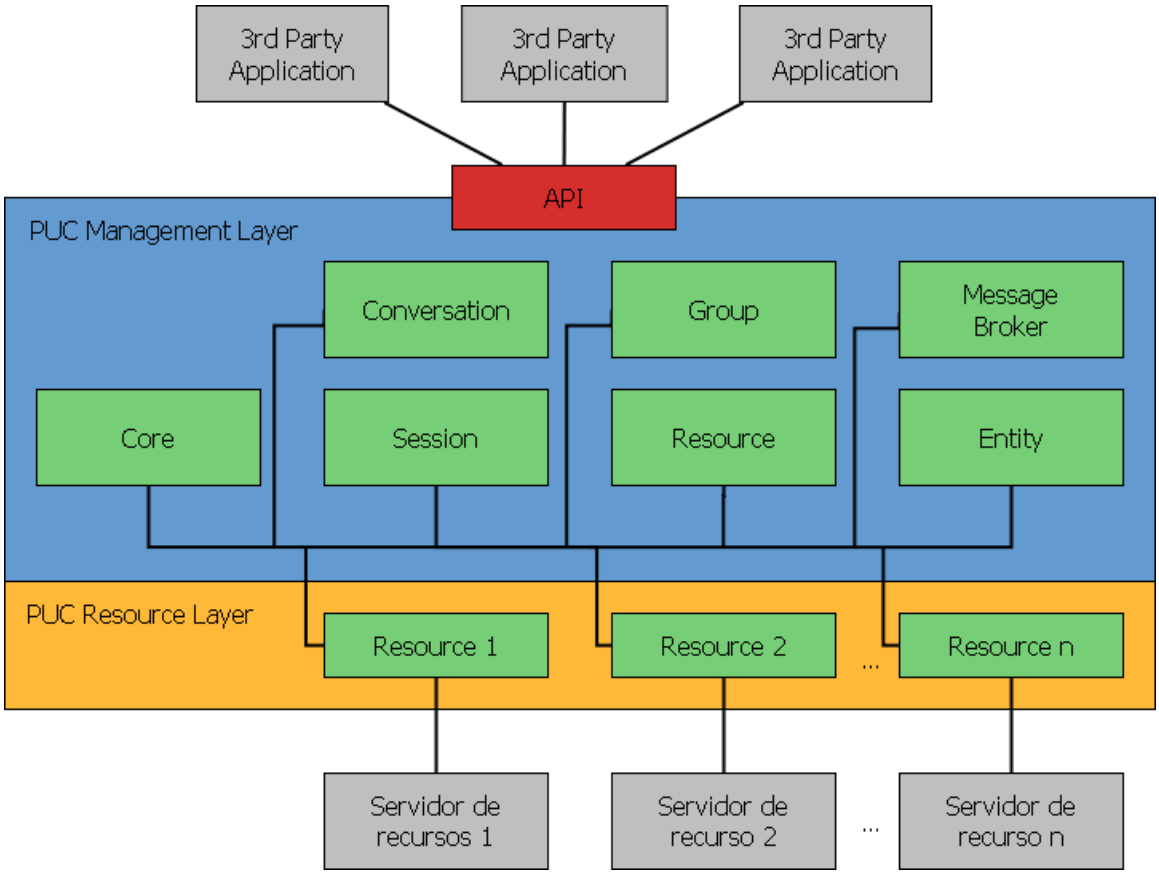

Figura 1: Arquitectura de alto nível do PUC

As camadas *Puc Management Layer* e *Puc Resource Layer* fazem parte daquilo que é considerado o *Core* da plataforma. A primeira destas duas camadas diz respeito aos diferentes *Managers* que têm como função a gestão de cada funcionalidade inerente à plataforma. Assim, o *Conversation Manager* gere as conversas, o *Group Manager* os grupos, o *Session Manager* as sessões e por aí fora. Outros dois módulos desta camada são o *Core* e o *Message Broker*. O primeiro faz a gestão das entidades e a respectiva adaptação à base de dados. É através deste módulo que são feitos os acessos à base de dados e se persistem os dados na mesma. O segundo oferece um serviço de troca de mensagens permitindo, assim, publicar e difundir diversos eventos que são usados pelas aplicações.

Na *Resource Layer* gerem-se os vários recursos disponíveis, que não são mais que representações na plataforma dos servidores externos que adicionam outras funcionalidades. Cada um destes servidores é gerido por um módulo que comunica com a camada superior (*Management Layer*) e estabelece a ligação de rede com o servidor externo. São exemplos de servidores externos o OpenMeetings, o Wave e o IP Windless.

O OpenMeetings é uma plataforma que tem como objectivo dar suporte a serviços de videoconferência, visualização colaborativa de diapositivos, *whiteboarding*, entre outros. É composta por diferentes componentes, como um servidor Flash, de código aberto, chamado Red5[10] e um servidor web, o Jetty[11] [12]. É através destes dois componentes que as aplicações cliente contactam a plataforma OpenMeetings de modo a poderem fazer uso dos recursos oferecidos pelo OpenMeetings, no que concerne às funcionalidades colaborativas já referidas. Associado, internamente, a este recurso, surge o conceito de "sala", como sendo um espaço virtual onde um determinado número de utilizadores se liga. Assim, "sala" corresponde ao conceito de sessão acima associado ao *Session Manager*. Doravante, sala e sessão assumem a mesma definição. O recurso do Wave adiciona à plataforma muitas das funcionalidades usadas no Google Wave[13], como a edição colaborativa de documentos ricos. O IP Windless é um Media Server que oferece diferentes funcionalidades, tais como *bridge* de áudio (entre múltiplas conversas), gravação da mesma, *speach to text* e interligação entre a rede IP e PSTN (rede telefónica fixa).

Por último, a camada superior é composta pelas aplicações construídas sobre a *framework* PUC. São exemplos destas o Formare[14] e Medigraf[15], respectivamente soluções de e-learning e telemedicina.

2.2. Trabalhos Relacionados

A plataforma PUC é um projecto interno à PT Inovação e que serviu de base à realização de diferentes Dissertações de Mestrado.

A primeira dissertação, realizada por Pedro Correia e denominada "A Framework for Collaborative Applications"[16] teve por principal objecto integrar o servidor de recurso OpenMeetings na plataforma e criar um mecanismo de comunicação entre este o seu controlador. Como acima foi mencionado, o OpenMeetings é um componente que terá como função facilitar o fornecimento de serviços de colaboração como *whiteboard*, video-conferência, entre outros.

Posteriormente, começaram a ser desenvolvidas outras quatro teses, sendo uma delas a que serve de suporte à escrita deste documento de Dissertação. As outras três visaram dotar a plataforma (PUC) de novas funcionalidades. Uma delas, realizada por Pedro Taborda[17], estudou a viabilidade, através da criação de um protótipo, de uma televisão, recorrendo à *set-top box*, se ligar à plataforma e participar em sessões colaborativas (nomeadamente na visualização de *slideshow*).

Outra dissertação, realizada por Bernardo Ferreira[18], visou tornar possível o projecto PUC oferecer funcionalidades idênticas às oferecidas pelo Google Wave[19] (como a comunicação e edição de documentos em tempo-real), adicionado como servidor de recurso, o servidor de protótipo da Google (FedOne)[20].

Por último, está ainda em desenvolvimento a dissertação "Mobi-Collab Aplicações de Colaboração para dispositivos móveis", realizada por João Sousa[21] e que visa a criação de widgets para a aplicação cliente, que ofereçam funcionalidades de controlo de sessão, como a escolha da funcionalidade colaborativa a usar, a sincronização de contactos do dispositivo móvel com a plataforma, entre outras.

3. Tecnologias Para o Desenvolvimento

Neste capítulo pretende-se apresentar um estudo das tecnologias relacionadas com o trabalho a desenvolver. Será apresentado o conceito de *widget* bem como alguns exemplos práticos da sua utilização. Serão ainda descritas algumas das tecnologias que poderão ser utilizadas para a realização do projecto proposto. Isto é, o desenvolvimento das já referidas componentes multimédia gráficas reutilizáveis que possam ser aproveitadas na criação de novas aplicações de colaboração nos terminais, nomeadamente em dispositivos móveis. Assim, nas próximas secções falar-se-à de aplicações de internet rica, Flex, Silverlight, JavaFX, OpenLaszlo e GWT. Depois de abordadas as diferentes tecnologias *per si*, será feita a comparação entre todas.

3.1. Widget

A noção de w*idget* surge normalmente como um componente de uma interface gráfica[22] que, usualmente, adiciona funcionalidades que não constam do programa "base". Embora actualmente existem widgets cada vez mais complexos e com mais funcionalidades, inicialmente, o conceito era associado a simples elementos de uma interface gráfica, como botões, barras, etc. Assim, pode-se dizer que um *widget*, por si só, não revela particular utilidade; é como componente de um programa maior que as suas capacidades podem ser realmente úteis ao permitirem o desenvolvimento de aplicações mais complexas. Deste modo, este tipo de componentes revela-se extremamente útil na sua reutilização por diferentes programas.

Actualmente, quase todos os sistemas operativos têm os seus próprios widgets, agora como extensões da interface ou do ambiente de trabalho. São exemplo disso os Windows Vista e Seven com a barra lateral e o OS X da Apple, com o *dashboard*. O Google, com o Google Desktop e o Yahoo! Widgets fornecem também vários destes aplicativos. Ao nível dos sítios da internet é também visível a introdução deste princípio, nomeadamente em blogs[23], através de "*web-widgets*".

Suponhamos querer desenvolver uma solução que permite a realização de telemedicina. Partindo desta base, o programador pode querer adicionar várias funcionalidades que tornem a solução mais viável, como por exemplo, videoconferência com partilha de documentos, possivelmente de raios-x ou outros exames e *whiteboard* que permita ao médico escrever algo que, eventualmente, torne mais fácil a compreensão da oralidade. É neste ponto que a utilização de *widgets* com estas funcionalidades se revelam muito úteis. O programador não precisa de se preocupar com os detalhes técnicos, como protocolos, ligações estabelecidas, etc., mas apenas adicioná-los à sua solução. Está, assim, criado um importante nível de abstracção.

O mundo dos *widgets* tem vindo a crescer e como consequência houve a necessidade de criar especificações para o desenvolvimento destes componentes. O Open Mobile Terminal Platform (OMTP) é um fórum criado por operadores de rede móvel para discutir as normas com os fabricantes de telemóveis e outros dispositivos móveis[24]. Em 2008, o OMTP lançou uma iniciativa chamada BONDI[25] cujo objectivo foi criar uma referência de implementação. Baseado nesta especificação, já existe, inclusive, um kit de desenvolvimento de software chamado BONDI SDK[26]. No entanto, para a realização desta dissertação, esta especificação não será considerada.

3.2. Aplicações de Internet Rica

Aplicações de internet ricas (do inglês RIA, *Rich Internet Applications*) são aplicações web cujas características, do ponto de vista do utilizador em tudo se assemelham às conhecidas aplicações de *desktop*. Começaram a ser faladas no ano de 2002 pela Macromedia[27] e o objectivo é trazer para o lado da interface do navegador web do utilizador todo o processamento gráfico, deixando a maior parte dos dados do lado do servidor da aplicação.

Tipicamente as RIAs consomem dados remotamente através de *web services,* processam-nos e mostram-nos, de forma visualmente apelativa, ao utilizador. O desenvolvimento deste tipo de aplicações tornou-se viável graças ao aumento do desempenho dos *browsers*, da banda larga e do poder computacional. Uma característica comum a este tipo de aplicações é que necessitam de um *client engine* que forneça ao navegador web a capacidade para suportar estas aplicações web. Assim é possível oferecer ao utilizador interfaces mais ricas que as tradicionais páginas web onde, inclusive, o cliente ganhe poder para manipulação de dados, não se limitando a fazer pedidos e envios para o servidor. Uma vez que o servidor não necessita de fazer todo o processamento e não está a receber constantemente pedidos é possível aliviar a carga do mesmo. Uma outra possibilidade é a de comunicação assíncrona permitindo que o cliente não fique bloqueado à espera da resposta do servidor.

Este tipo de aplicações tem, no entanto, também algumas restrições. Se o utilizador tiver desactivado a execução de scripts no seu navegador web, a RIA poderá não correr ou correr de forma limitada. Um outro ponto prende-se com a velocidade de processamento do lado do cliente. É frequente a utilização de Javascript que, como é sabido, é uma linguagem que necessita ser interpretada pelo pelo navegador web, ficando, assim, dependente da velocidade do lado do cliente. Outro aspecto menos positivo é o tempo que a aplicação pode demorar a carregar, uma vez que não é instalada e tem que ser descarregada para o navegador web. É possível optimizar este processo fazendo uso da *cache*.

Algumas tecnologias têm sido desenvolvidas para suportar mais facilmente o desenvolvimento deste tipo de aplicações (RIA). Nas secções seguintes descrevem-se algumas destas.

3.3. Flex

Flex é uma tecnologia lançada pela Adobe cujo objectivo é o desenvolvimento de aplicações de internet rica[28]. A sua versão, estável, mais actual é a 3. A Adobe disponibiliza um ambiente integrado de desenvolvimento chamado Flex Builder. Esta é uma plataforma baseada no conhecido Eclipse e que tende a facilitar a programação. Ao nível das linguagens de programação, esta tecnologia é desenvolvida em MXML e ActionScript. MXML é uma linguagem de marcação baseada em XML, inicialmente criada pela Macromedia (empresa comprada pela Adobe). Por sua vez, ActionScript é uma linguagem orientada a objectos, baseada em ECMAScript, que é executada na máquina virtual denominada ActionScript Virtual Machine. Esta máquina virtual encontra-se no *plugin* Adobe Flash Player e no ambiente Adobe Air. O primeiro corre nos navegadores web, já o segundo permite que aplicações construídas em Flex possam ser executadas como qualquer outra aplicação; isto é, sem recurso ao navegador web.

Actualmente, a versão estável para o Adobe Flash Player é a 10.1. Esta versão do conhecido *plugin* para navegadores web, que permite a execução de ficheiros Flash, tem como grande vantagem o facto de estar a ser incorporado em dispositivos móveis como o Android e Symbian S^3 e ser disponibilizado na maioria dos dispositivos fixos; segundo a Adobe, mais de 98%[29] dos *desktops* com Windows, Mac e Linux têm o *plugin* instalado.

Open Screen Project é um projecto cujo objectivo é oferecer uma interface consistente entre dispositivos móveis e computadores pessoais[29]. Para que tal aconteça a Adobe tomou algumas medidas, tais como, abolir as taxas de licenciamento para o Adobe Flash Player e Adobe Integrated Runtime, removeu as restrições para o uso dos formatos Shockwave Flash (SWF) e Flash Video (FLV), disponibilizou a API do Flash e

publicou o protocolo Flash Cast e Action Message Format (AMF) que permite que aplicações Flash recebam informação de bases de dados remotas.

Neste projecto, são vários os parceiros existentes[30], entre os quais as principais empresas de telemóveis como a Nokia, Motorola, Palm, HTC, Google, entre outras. O intuito é que a nova versão do Adobe Flash Player, a 10.1, seja compatível com a grande maioria dos dispositivos móveis e assim, abranger o maior número possível de dispositivos, móveis e não só, com a tecnologia Flash. Segundo a Adobe, no início de 2010 serão disponibilizadas versões compatíveis com Windows Mobile, webOS, Android e Symbian[31]. A compatibilidade com o iPhone não é garantida, contudo a Adobe irá lançar um produto designado por Flash Professional CS5 que permitirá portar aplicações Flash para código compatível com o iPhone.

3.4. Silverlight

O Silverlight, cuja versão actual é a 4, é uma tecnologia desenvolvida pela Microsoft com o objectivo de, à semelhança da acima descrita, criar um ambiente propício ao desenvolvimento de aplicações de internet rica. Usando Silverlight, as interfaces do utilizador são declaradas em XAML[32] e programadas usando um subconjunto da estrutura .NET. XAML, cuja sigla significa eXtensible Application Markup Language, é uma linguagem declarativa, baseada em XML, da Microsoft para facilitar a criação de interfaces para o utilizador.

Actualmente, a Microsoft disponibiliza o Silverlight apenas para as plataformas Windows e Mac[33]. Contudo, um projecto designado por Mono, liderado pela empresa Novell, desenvolveu uma implementação de código aberto da tecnologia Silverlight, chamado Moonlight[34].

Devido ao facto de este produto ser ainda novo e, oficialmente, não ser multi-plataforma a sua taxa de penetração é ainda baixa.

No que toca à compatibilidade com os dispositivos móveis, segundo a Microsoft, o Silverlight será disponibilizado em Windows Mobile e Nokia S60[35].

3.5. JavaFX

JavaFX é uma plataforma, criada pela Sun Microsystems (agora Oracle), cujo objectivo é, mais uma vez, desenvolver aplicações de internet rica. A versão actual é a 1.3 e goza da transversalidade ao nível de sistemas operativos; tem suporte para Windows, Mac e Linux , pois apenas necessita do Java Runtime Environment (JRE) instalado para poder correr aplicações produzidas em JavaFX.

Ao nível da programação a Sun disponibiliza um *plugin* para o conhecido ambiente de desenvolvimento integrado, NetBeans. Este *plugin* permite fazer uso, de forma mais prática, do JavaFX 1.2 SDK. Este kit de desenvolvimento de software (SDK) incorpora o compilador JavaFX e o ambiente de execução, entre outras coisas[36], tendo em vista o desenvolvimento de aplicações ricas em internet para *desktop*, navegadores web e dispositivos móveis. A linguagem utilizada para programar, JavaFX Script, é uma linguagem de *script*, orientada a objectos, com métodos (chamados operações e funções) e atributos, à semelhança de Java[37].

No que diz respeito aos dispositivos móveis, a Sun afirma ser possível correr a versão Mobile do JavaFX em praticamente todos eles. Isto acontece porque esta tecnologia corre directamente em cima de Java ME. Ora, Java ME (Micro Edition) está presente na grande maioria dos telemóveis. Segundo a Sun, 8 em cada 10 telemóveis, em 2008, foi comercializado com este componente instalado[26].

3.6. OpenLaszlo

O OpenLaszlo é uma plataforma de código aberto para desenvolvimento e distribuição de aplicações de internet rica[38]. Actualmente na sua versão 4.7, as aplicações OpenLaszlo necessitam apenas do Adobe Flash Player instalado, do lado cliente, para poderem ser executadas[39]. Dado necessitarem apenas deste requisito, à semelhança das aplicações produzidas em Flex, estas também poderão ser executadas em

navegadores web que corram sobre Windows, Mac e Linux. Assim, e gozando da grande popularidade do Adobe Flash Player, é possível disponibilizar estas aplicações em mais de 98% dos computadores ligados à internet[40].

Como linguagem de programação é utilizada LZX[39]. LZX é uma linguagem baseada em XML e Javascript que permite um processo de desenvolvimento baseado em texto.

Orbit é um projecto que juntou Sun e a OpenLaszlo[41] e cujo objectivo seria permitir que aplicações desenvolvidas em LZX pudessem correr utilizando a plataforma Java ME. Contudo, o projecto parece ter sido descontinuado uma vez que a última actualização visível data de há 3 anos.

Uma vez que as aplicações de OpenLaszlo são compiladas para correm sobre Adobe Flash Player, é possível que possam vir a ser utilizadas em dispositivos móveis, sensivelmente aquando do lançamento deste *plugin* nos diferentes dispositivos.

3.7. GWT – Google Web Toolkit

Analisadas as tecnologias que permitem desenvolver as funcionalidades acima referidas, estudou-se também uma que fosse baseada apenas em tecnologias web e independentes do flash. Uma vez que para a realização de *download*, *upload*, listagem e remoção de ficheiros são apenas necessárias as capacidades de um *browser*, no que toca a ligações HTTP, HTML e, possivelmente, JavaScript, optou-se por utilizar uma tecnologia que respondesse a estas necessidades e que, facilmente, permitisse a criação de widgets sem que fosse requerida uma grande curva de aprendizagem. Uma vez que a linguagem de programação utilizada no GWT é JAVA e a compilação do código é feita para tecnologias web, optou-se pelo uso desta ferramenta.

Google Web Toolkit é uma ferramenta de código aberto, desenvolvida pela Google que permite a programadores desenvolver aplicações baseados no *browser*[42]**.** O GWT oferece um SDK, juntamente com um *plugin* para o Eclipse, que permite

programar a aplicação utilizando a linguagem Java e, no final, exportá-la como HTML e JavaScript. Uma vez que o resultado final é uma aplicação AJAX, os widgets aqui desenvolvidos tenderão a correr em todos os *browsers*[43]. Segundo a Google, produtos como o Google Wave e AdWords são desenvolvidos utilizando esta ferramenta.

3.8. Comparação

Feita uma descrição de algumas das tecnologias existentes, é necessário agora fazer uma análise comparativa sobre aquelas que competem para as mesmas funcionalidades. Isto é, Flex, Silverlight, JavaFX e OpenLaszlo.

Tendo em conta o objectivo inicialmente traçado de criar uma estrutura que permita a maior ubiquidade possível, podendo estar disponível no mais largo espectro de dispositivos existentes, à partida, a tecnologia Silverlight é a que reúne menos vantagens. Recordemos que, oficialmente, esta plataforma está apenas disponível para Windows e Mac, ao passo que as outras têm versões para Windows, Mac e Linux. Do ponto de vista do desenvolvimento isto pode ser um constrangimento considerável. Ainda como consequência da não disponibilização de versões oficiais para os três sistemas operativos mais conhecidos e devido ao facto do Silverlight ser um produto relativamente recente, o número de computadores com esta tecnologia instalada é diminuto em relação às outras plataformas.

Visto que o âmbito deste trabalho se foca mais nos dispositivos móveis, esse é um ponto essencial para a escolha da tecnologia a usar. É fundamental que esta funcione no maior número de sistemas operativos móveis possível. Assim, o Silverlight passará imediatamente para último lugar, uma vez que, inicialmente, a Microsoft prevê apenas o lançamento para Windows Mobile e Nokia S60.

A Gartner[44], empresa líder em assessoria de tecnologias de informação elaborou um estudo sobre a evolução da penetração dos diferentes sistemas operativos móveis no mercado que se traduz no seguinte gráfico:

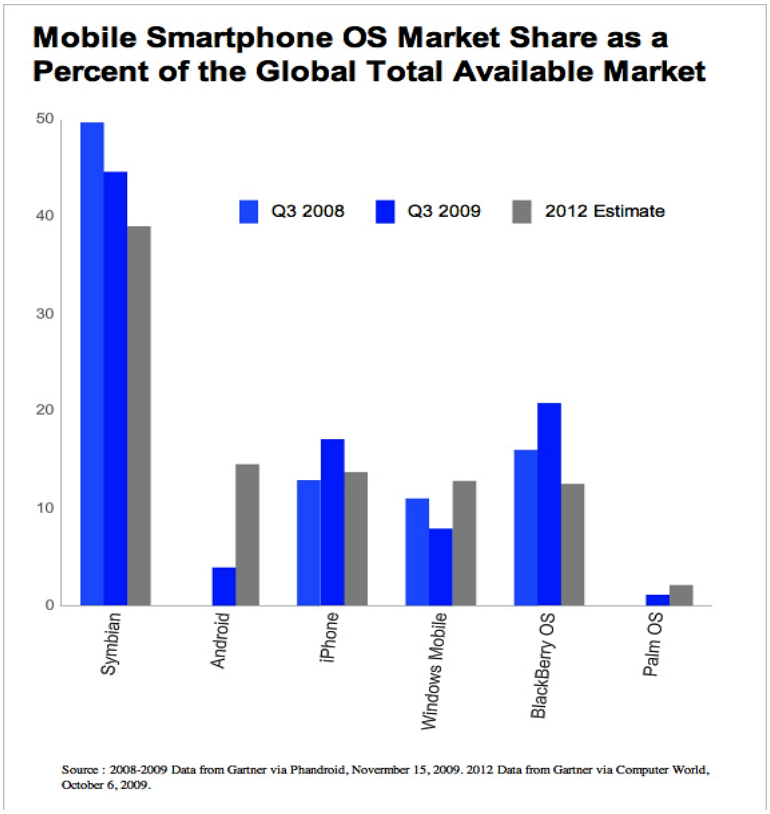

Figura 2: Gráfico sobre a evolução dos sistemas operativos móveis no mercado

É possível constatar que se estima que em 2012 o Symbian continue a ser o sistema operativo mais comum e o Windows Mobile o quarto. No entanto, a quota de mercado do Android, em claro forte crescimento, não deve ser desprezada.

Ainda assim, e de modo a podermos concluir algo suficientemente sólido, será necessário introduzir, a nível comparativo, de que maneira estas tecnologias suportam, ou não, a implementação das capacidades apresentadas na introdução: videoconferência, *remote desktop*, *desktop sharing*, visualização de diapositivos e *whiteboard*. Apesar de, pelas razões acima invocadas, a tecnologia Silverlight estar excluída, será incluída neste nível de comparação uma vez que foi, também, objecto de estudo.

No que toca à videoconferência, apenas o JavaFX não permite a sua implementação uma vez que esta tecnologia não suporta a captura de som do microfone e imagem via webcam[45]. Qualquer outra das tecnologias estudadas permite a implementação desta funcionalidade.

Uma vez que a funcionalidade de *Desktop Sharing* necessita de permissões para a gravação do ambiente de trabalho do utilizador é necessário utilizar Java Applet em conjunto com qualquer das tecnologias abordadas. Por si só, o Flash não tem permissões para acesso ao ambiente de trabalho e fazer a sua gravação[46]. O mesmo acontece com a tecnologia OpenLaszlo e JavaFX. No que concerne à tecnologia Silverlight não me foi possível apurar com precisão essa possibilidade, no entanto, esta plataforma permite a realização de *Remote Desktop,* o que poderá indiciar essa capacidade. Uma vez ultrapassado este obstáculo, a gravação que vai sendo feita é enviada em tempo real para os restantes participantes.

Remote Desktop, como descrito acima é a capacidade que um programa tem em se ligar a outro computador e, de algum modo, o poder controlar. Tal é possível através de Flex e Silverlight. Faz prova dessa aplicabilidade no Flex o projecto FlashLight-VNC[47]. Este projecto consiste num cliente de VNC escrito em Flash e distribuído sob a licença LGPL. Contudo é requerido que do lado de quem emite esteja a correr um servidor de VNC.

Quanto à tecnologia Silverlight, existe um *Proof of Concept*[48] criado por Brendan[49] e que ilustra a possibilidade da criação de um cliente de Silverlight.

Relativamente à tecnologia JavaFX não me foi possível confirmar tal possibilidade, no entanto é possível fazer utilizando as Applets de Java, como demonstra o cliente do conhecido produto RealVNC[50].

A visualização colaborativa de diapositivos é possível de ser aplicada com qualquer uma das tecnologias em estudo. Como referido anteriormente, é enviado o documento contendo os diapositivos para o servidor (um .ppt, por exemplo) que se encarrega de converter em imagens e sincronizar com os restantes dispositivos. Deste modo, do lado do terminal apenas tem que ser assegurada a possibilidade da visualização de imagens e, neste caso, qualquer das tecnologias o permite.

A criação de um *whiteboard* é também possível fazendo uso de qualquer umas das tecnologias apresentadas. Neste caso, quando um utilizador escreve algo no quadro virtual, são enviadas para os restantes utilizadores a ferramenta utilizada e as coordenadas do desenho (recta, ponto, etc). Nos outros terminais é apenas necessário reproduzir o que foi recebido através de desenhos geométricos. Para tratar da conflitualidade que possa advir da utilização simultânea dos mesmo "objectos" as operações são sincronizadas. Deste modo evitam-se os conflitos de edições simultâneas e define-se implicitamente uma ordem, não sendo necessário definir qualquer outro tipo de prioridade.

Em baixo segue o esquema comparativo em forma de quadro:

Funcionalidade	**Video-conferência**	**Desktop Sharing**	**Remote Desktop**	**Visualização de Slideshow**	**Whiteboard**
Descrição	"Chamada" com áudio e vídeo	Partilha sem interacção com desktop remoto	Partilha e interacção com desktop remoto	Visualização colaborativa de diapositivos	Espaço único e partilhado de apresentação de um "quadro"
Tecnologia					
Flex	Sim*[I]	Sim*[1], utilizando, por exemplo, Java applet*[II]	Sim*[III]	Sim*[2]	Sim*[5]
Silverlight	Sim*[IV]	Sim*[3]	Sim*[V]	Sim*[2]	Sim*[VI]
JavaFX	Não*[VII]	Sim (é possível através de Java applet)	Sim (é possível através de Java applet)	Sim	Sim*[5] (javafx.scene.shape)
OpenLaszlo	Sim	Sim		Sim*[2]	Sim

*I – [51] ; *II – [52] ; *III – [47] ; *IV – [53] ; *V – [48] ; *VI – [54] ; *VII - [45]
*1 - por questões de segurança não é permitido com o flash fazer desktop recording.
*[2] - por exemplo, através do Red5 (envio de imagens para os restantes utilizadores)
*[3] - se é possível Remote Desktop, será Desktop Sharing.
*[4] - sim, se do lado de quem emite estiver a correr um VNC Server
*[5] - são enviadas as coordenadas e a ferramenta usada. Conforme o enviado, a aplicação faz o desenho.

Tabela 1: Quadro comparativo entre diferentes tecnologias vs. funcionalidades multimédia

Dadas as características acima descritas e analisando os prós e contras é possível concluir que ter-se-à que optar ou por Flex ou OpenLaszlo, uma vez que qualquer uma das outras tecnologias apresenta limitações em pontos fundamentais. No caso do Silverlight já foi referida a pouca implementação ao nível dos dispositivos móveis, o que

a faz excluída à partida. Quanto ao JavaFX, o facto de não permitir a captura de som do microfone e imagem da webcam torna inviável a sua utilização no contexto pretendido.

Relativamente às outras duas tecnologias, Flex e OpenLaszlo, a segunda apresenta mais inconvenientes que a primeira. Uma vez que o OpenLaszlo utiliza a tecnologia Flash, estará sempre dependente deste e compilará, muitas vezes, para versões diferentes da actual, especialmente se o lançamento da última versão do *plugin* de Flash tiver sido recente. Optando pelo Flex, garantimos conseguir utilizar sempre as versões mais recentes e, assim, tirar partido das melhorias e novas funcionalidades que vão sendo introduzidas.

Um outro ponto que ajuda a escolha a recair na tecnologia Flex é a possibilidade que este tem em serem criados *Flex Library Projects* (SWC)[55]. Estes projectos permitem criar bibliotecas que podem ser partilhadas e distribuídas por outros criadores de programas. Para que tal aconteça, é apenas necessário adicionar ao caminho das bibliotecas do projecto Flex o ficheiro SWC. Esta é uma característica que não está disponível utilizando a tecnologia OpenLaszlo[56].

3.9. Protocolos

Para alguns tipos de aplicações, caso da videoconferência, existe a necessidade de recorrer a protocolos para transferência de dados (*stream*) como áudio e vídeo. Neste sentido foram estudados alguns protocolos que pudessem responder a essa mesma necessidade.

Uma vez feita a opção de utilização de Flex, torna-se essencial o estudo de protocolos que permitam implementar as funcionalidades acima descritas. Deste modo, é necessário por um lado abordar protocolos que permitam estabelecer uma sessão e por outro, outros protocolos que permitam a troca dos dados pretendidos. Um exemplo desses dados são os de multimédia, como *streams* de áudio e vídeo para videoconferência. A implementação dos mesmo já está feita e encontra-se fora do âmbito desta dissertação.

Contudo, penso ser necessário o seu estudo para uma correcta visão global e escolha dos protocolos a serem usados.

Como ponto de partida para escolha dos protocolos a estudar, procurei saber quais os mais comuns e, fundamentalmente, quais os compatíveis com a tecnologia a adoptar, neste caso o Flex. Tive ainda em conta que no projecto em que está inserida esta dissertação, a arquitectura utilizada é a de cliente-servidor. Isto é, é o cliente que comunica sempre com o servidor e não directamente com outro cliente. Este tipo de arquitectura evita que um dado cliente tenha que utilizar um serviço de descoberta para comunicar com outro. Todas as comunicações passarão sempre por um servidor que se encarregará da troca de mensagens.

Posto isto, nas secções seguintes serão, então, abordados os seguintes protocolos para estabelecimento de sessão e para troca de dados: RTMP e SIP/RTP.

3.9.1. RTMP

Real Time Messaging Protocol (RTMP) é um protocolo desenvolvido pela Adobe, empresa responsável pelo Flex/Flash, para streaming de áudio, vídeo e dados entre o Flash Player e um servidor. O protocolo apresenta 3 variações. Na versão "base" o protocolo trabalha sobre TCP e usa a porta 1935. Nas outras duas versões, os dados são encapsulados em pacotes HTTP ou HTTPS.

Uma vez que RTMP foi concebido para trabalhar com RTMP Chunk Stream[57], este será de seguida abordado.

RTMP Chunk Stream é um protocolo de nível aplicação desenhado para trabalhar com o RTMP, apesar deste suportar qualquer outro protocolo que envie uma *stream* de mensagens. É, no fundo, um protocolo que permite o envio de mensagens partidas em blocos (chunks), possivelmente mais pequenos.

Os dois protocolos, em conjunto, podem ser usados numa grande variedade de aplicações áudio-video tais como, pessoa-a-pessoa, em grupo, *broadcasting* ou *video-on-demand*.

RTMP Chunk Stream pode ser usado com TCP (um protocolo fiável na entrega de pacotes) garantido, por isso, que as mensagens são entregues pela ordem correcta, baseado em estampilhas temporais.

O *Hanshake*, o estabelecimento de uma sessão é um ponto importante em qualquer protocolo. Neste caso, o protocolo para estabelecer uma sessão consiste no envio de três chunks (fragmento de uma mensagem) de tamanho fixo pelo cliente e outros tantos pelo servidor. O objectivo é a definição da versão do protocolo a usar (que tem que ser, forçosamente, a mesma) e de uma época temporal. Isto é, será o valor tido como inicial na comunicação. Todas as outras estampilhas temporais deverão ter um valor superior. São ainda enviados pacotes cujo significado é apenas de *acknowledge* (ack). A partir deste ponto ambas as partes, cliente e servidor, estão em condições de proceder à troca das mensagens que se pretende.

3.9.2. SIP

SIP, Session Initiation Protocol, é um protocolo de aplicação desenhado para criar, modificar e terminar sessões[58] entre um ou mais participantes. A sua especificação mais actual é a RFC 3261 do IETF (Internet Engineering Task Force, organismo responsável pela administração e desenvolvimento de mecanismos no seio da Internet) [59]. Uma vez que o protocolo tem como objectivo apenas controlar sessões e tornar possível a comunicação, é necessário associar outro protocolo que torne possível comunicação em si. Isto é, o estabelecimento, mudança ou término da sessão é independente do tipo de media ou aplicação que será usada.

Um dos protocolos frequentemente usado com o SIP é o RTP (Real-time Protocol). Este é usado no transporte de dados multimédia, como áudio, vídeo ou texto, em tempo real, tornando possível codificar e dividir os dados em pacotes e transportá-los pela Internet. Tal acontece porque o SIP necessita de outros protocolos que garantam o transporte. Com o SIP associa-se também o SDP (Session Description Protocol), utilizado nas mensagens SIP para descrever o estabelecimento das sessões multimédia.

Existem 5 características importantes no protocolo SIP que explicam a sua utilidade e objectivos de utilização: localização do utilizador a comunicar; determinação da disponibilidade do utilizador; determinação das capacidades do *media* e os parâmetros a serem usados na comunicação; estabelecimento dos parâmetros de sessão entre os utilizadores; gestão da sessão que inclui transferir e terminar a sessão, alterar os parâmetros da sessão e a invocação de serviços.

As mensagens SIP são idênticas às mensagens HTTP (RFC 2068). Grande parte da sintaxe usada nos cabeçalhos e dos códigos das mensagens HTTP são os mesmos usados pelo SIP. A título de exemplo, o código 404 usado no protocolo HTTP numa situação de erro em que não se encontra o URL pretendido, é também usado pelo SIP com a mesma finalidade.

Uma outra semelhança entre o protocolo SIP e o protocolo HTTP, reside no facto de ambos serem do tiipo pedido-resposta (*request-response*). Assim, temos apenas dois tipos de mensagens no protocolo SIP, as de pedido e as de resposta, definidas na norma RFC 3261[59]. As mensagens existentes têm como objectivo registar um utilizador, enviar convites para o estabelecimento de uma sessão, rejeitar o início da mesma, terminar uma sessão em concurso e as respectivas mensagens de resposta.

3.9.3. RTP/RTCP

RTP, do inglês Real Time Protocol, é definido pelo RFC 3550[60] como um protocolo utilizado para o transporte de dados contínuos (*stream*), em tempo real, numa ligação ponto-a-ponto, como áudio ou vídeo. É também possível ser usado não apenas em ligações ponto-a-ponto mas também para múltiplos destinatários, utilizando para isso o endereço de IP reservado para grupos *multicast*. Podemos, então, resumir o RTP como tendo como objectivo fornecer um meio uniforme para transmitir dados sujeitos a constrangimentos de tempo real (áudio, vídeos,…) e cujo papel principal consiste em garantir que cada utilizador recebe os pacotes pela ordem correcta ainda que a rede subjacente altere a ordem dos mesmos. Uma vez que estamos a falar de dados

multimédia, a ordenação temporal correcta é mais importante que a fiabilidade. Por exemplo, a perda de um segundo de áudio é facilmente colmatada através de algoritmos de correcção de erros[61]. Assim, a garantia da correcta ordenação dos pacotes tem que ser feita pelo RTP porque, apesar do TCP ser o protocolo de transporte por excelência daquele[62], é muitas vezes usado o UDP para evitar a latência introduzida pelo TCP no estabelecimento da ligação e detecção de erros. Referira-se ainda que diferentes tipos de dados serão enviados em diferentes sessões de RTP mesmo que façam parte da mesma comunicação. Por exemplo, numa videoconferência são transmitidos dois tipos de dados (áudio e vídeo) e, assim, os pacotes de áudio serão transmitidos por uma sessão RTP enquanto os pacotes de vídeo serão transmitidas por outra sessão RTP diferente.

O RTP pretende identificar o tipo de informação transportada, acrescentar estampilhas temporais e números de sequência à informação transportada e controlar a chegada ao destino dos pacotes. Contudo, por si só o protocolo não fornece nenhum mecanismo para garantir a entrega dos dados ou qualidade do serviço, relegando isso para a camada inferior.

Uma vez que o protocolo RTP não reserva recursos, nem garante qualidade do serviço, é normalmente usado em conjunto com o protocolo RTCP (RTP Control Protocol). Este, é utilizado para monitorar a qualidade do serviço e para transmitir informações sobre os participantes numa sessão em curso.

O protocolo RTCP baseia-se em transmissões periódicas de pacotes de controlo por todos os participantes da sessão. É um protocolo de controlo de fluxos RTP que fornece estatísticas e informação sobre uma ligação RTP. Assim, a principal função do RTCP é fornecer *feedback* sobre a qualidade de serviço, enviando periodicamente informações estatísticas para os participantes da sessão.

O RTP e o RTCP são protocolos que se situam a nível da aplicação e utilizam os protocolos subjacentes de transporte TCP ou UDP. No entanto, a utilização de RTP/RTCP faz-se geralmente sobre UDP. Cada um dos protocolos usa uma porta diferente; o RTP utiliza a porta par e o RTCP a porta ímpar imediatamente acima.

O cabeçalho de um pacote RTP contém alguns campos importantes e dos quais se destaca a versão do protocolo a usar, a definição do número de sequência que

posteriormente será sempre incrementado e a estampilha temporal que permite sincronização e saber a correcta ordem dos pacotes. Em anexo é apresentado o esquema completo do cabeçalho de um pacote RTP e todos os seus campos.

Do cabeçalho do pacote RTCP há a destacar o campo que identifica o número de pacotes perdidos na transmissão e o *Interarrival jitter* que é uma estimativa do intervalo de tempos de um pacote de dados RTP, medido com o *timestamp* e que está sob a forma de um número inteiro. É, assim, o tempo relativo de trânsito entre dois pacotes de dados. Em anexo são apresentados todos os campos constantes no cabeçalho de um pacote RTCP.

3.9.4. Comparação de Protocolos

O protocolo RTMP poderá ser comparado com a conhecida solução SIP/RTP. Ambos apresentam vantagens e desvantagens.

O SIP apresenta a vantagem de permitir a procura do utilizador a comunicar. Tal funcionalidade não se verifica no RTMP. Para além disso, o *overhead* introduzido pelo último, no *Handshake*, é substancialmente maior devido à quantidade de mensagens trocadas e aos campos delas constantes.

Por outro lado, uma das grandes diferenças entre RTMP e RTP reside no facto de o último relegar noutros protocolos o controlo da comunicação. Por exemplo, o protocolo Media Gateway Control Protocol (MGCP) usa o Session Description Protocol (SDP) para a negociação das *streams* de media a serem transmitidas e o RTP apenas para o "empacotamento" das *streams*.

Um outro ponto a favor do RTMP prende-se com o facto de este já ter previsto o *Handshake* da sessão. O RTP, por seu lado, necessita de se associar a outros protocolos para alcançar esse objectivo, como por exemplo o SIP.

Uma vez que o projecto a realizar se enquadra numa estrutura que inclui, como recurso, um servidor Flash, poder-se-à tornar intuitivo escolher o protocolo RTMP, da Abode (empresa proprietária do Flash). Adicionalmente, usando, como é o caso, um

esquema cliente/servidor, uma das vantagens do SIP que residia na possibilidade de encontrar o utilizador que se pretender ligar é relegada para um plano secundário. Numa situação destas, é o cliente que inicialmente contacta o servidor e sabe onde ele se encontra, portanto, o problema da localização do servidor a contactar não se põe.

Para além das características acima mencionadas há ainda a referir que o protocolo RTP foi apenas desenvolvido para trabalhar com áudio e vídeo. Assim, funcionalidades como *whiteboard*, *remote desktop* ou mesmo a visualização colaborativa de diapositivos ficariam hipotecadas.

Na solução proposta pela PT Inovação é ainda necessário lidar com eventos enviados pelo servidor e, nessa situação, o recurso ao protocolo RTMP com chamadas remotas torna a implementação bem mais fácil. Neste caso, é necessário definir qual o formato de codificação dos objectos usado. Tipicamente, é utilizado o formato AMF.

AMF, Action Message Format, é um formato compacto binário, usado para serializar objectos do tipo ActionScript.[46] É usado normalmente na troca de dados entre uma aplicação Adobe Flash e um serviço remoto. A primeira versão, AMF0, suporta o envio de objectos complexos por referência, o que ajuda a prevenir o envio de instâncias redundantes. A nova versão, AMF3, surgiu aquando do lançamento do ActionScript 3.0 e trouxe algumas melhorias em relação à anterior versão bem como o suporte para novos tipos de dados introduzidos com a versão 3.0 do ActionScript.

Além de serialização de tipos ActionScript, o AMF pode ser utilizado nas invocações assíncronas de serviços remotos. Uma estrutura simples de mensagem é usada para enviar um conjunto de pedidos para um ponto remoto. O formato desta estrutura é AMF0.

4. Desenho da Solução

Enquadrado na plataforma já descrita (PUC) e tendo em conta os requisitos necessários à elaboração do projecto, poder-se-à sintetizar, novamente, o objectivo principal do mesmo como sendo o estudo e desenvolvimento de componentes gráficas multimédia reutilizáveis para o desenvolvimento de novas aplicações de colaboração para os diferentes terminais do utilizador. Esses componentes deveriam permitir a partilha de recursos multimédia tais como videoconferência,, visualização colaborativa de diapositivos, *whiteboard*, bem como a possibilidade de partilha de ficheiros através de *upload* e *download*. Apesar do foco ser a sua funcionalidade nos dispositivos móveis, foi, também, necessário escolher uma tecnologia que pudesse abranger o maior número de sistemas operativos não móveis. Deste modo, seria encontrada uma solução que permitisse a maior universalidade possível ao nível dos terminais móveis e fixos.

Apesar da abstracção que se pretende impor entre os recursos e as aplicações existe, em alguns casos, a necessidade de interacções directas entre o recurso e a aplicação, como é o caso do OpenMeetings. Uma vez que este é baseado num servidor flash (Red5), as aplicações que fizessem uso das funcionalidades por ele oferecidas teriam que comunicar directamente com o referido recurso, não sendo possível fazê-lo apenas através da camada imediatamente abaixo da aplicacional como sugere a Figura1. Por outro lado, é necessário assegurar que a camada de *Management Layer* continua com o controlo da sessão apesar da aplicação não se ligar directamente a ela.

De modo a dar resposta a estes requisitos foram criados módulos que funcionam como pequenas “aplicações” (widgets) que podem ser incorporadas na aplicação principal do cliente e cujo intuito é disponibilizar a funcionalidade associada ao recurso

sem que quem desenvolve a aplicação principal necessite conhece-lo. Assim, garante-se a desejada abstracção para o nível aplicacional.

A cada diferente funcionalidade oferecida pelo recurso OpenMeetings fez-se corresponder um widget diferente. Assim, cada módulo pretende ser independente e provido de toda a lógica inerente à funcionalidade que diz respeito. Uma vez desenvolvidos, as diferentes aplicações podem usar os widgets de acordo com as ferramentas que pretendem apresentar ao utilizador. A incorporação destes componentes pretende-se que seja o mais transparente e simples possível.

Eis o esquema do que em cima foi explanado:

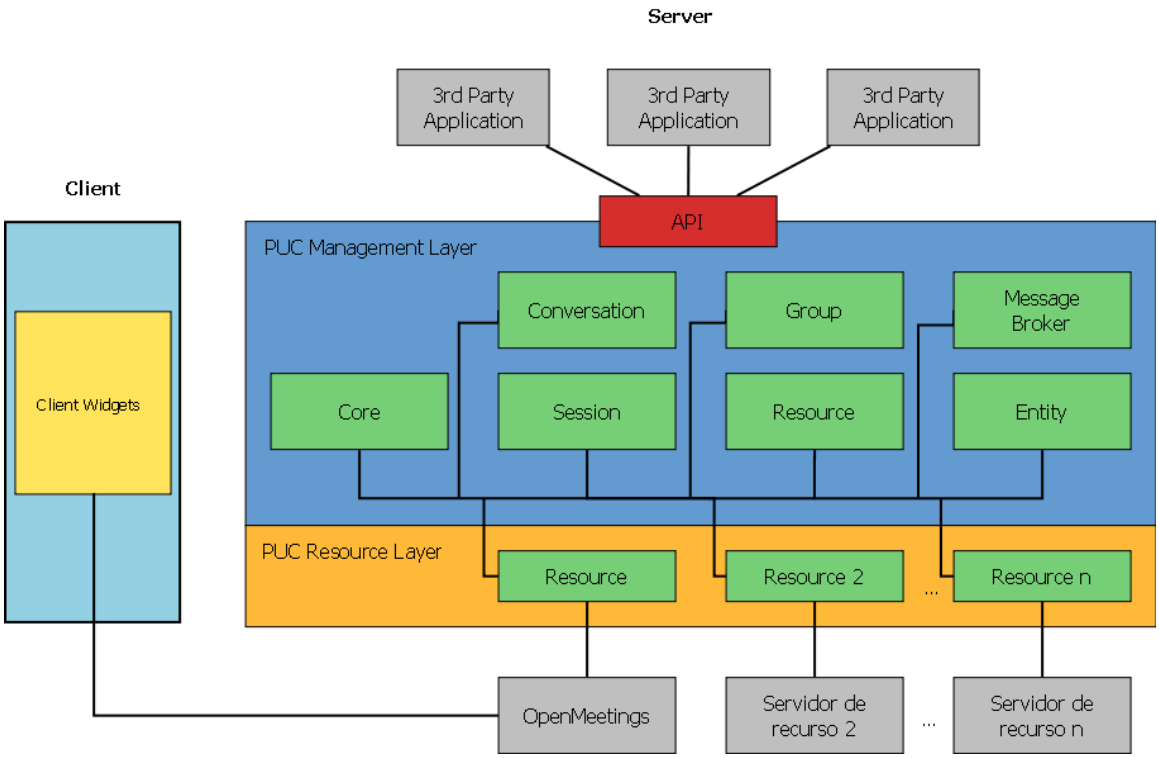

Figura 3: Arquitectura de alto nível do PUC em comunicação com os widgets de cliente

O componente designado por Client Widgets pretende representar os diversos widgets já abordados. Como se pretende demonstrar, os widgets fazem parte da componente cliente e comunicam directamente com o recurso.

Contudo, e como referido anteriormente, é necessário que a camada de Management Layer continue com o controlo da sessão. Para isso, criou-se um mecanismo que permitisse essa comunicação. Tal solução passou pela implementação de um sistema de notificações entre o OpenMeetings e o controlador do referido recurso e entre este os enablers da camada Management Layer. Uma outra razão para a necessidade do uso destas notificações prende-se com o facto de diferentes terminais poderem, potencialmente, partilhar a mesma funcionalidade. A título de exemplo, suponhamos que um utilizador, ligado directamente ao OpenMeetings, inicia uma apresentação. Caso não existisse o referido esquema de notificações, apenas os clientes ligados ao OpenMeetings poderiam partilhar a mesma apresentação, uma vez que o próprio servidor avisa todos os utilizadores presentes na sala das alterações que vão ocorrendo (como o início da apresentação, mudança de *slide* ou fim da mesma). No entanto, utilizando a solução sugerida é possível que um utilizador cujo terminal seja uma televisão (*set-top box*) e esteja ligado à plataforma (PUC) que não através do OpenMeetings veja a mesma apresentação. Tal é conseguido porque, com esta abordagem, é possível que a aplicação (cliente) recebe as notificações e possa assim acompanhar a apresentação, mesmo não estando directamente ligada ao OpenMeetings. Mais se pode ler sobre o processamento necessário na set-top box, na dissertação "PLAY: Terminal IPTV para Visualização de Sessões de Colaboração Multimédia"[17].

O contexto das comunicações com o OpenMeetings é sempre uma sala. A noção de sala, como referido anteriormente, é um espaço virtual onde um ou mais utilizadores se podem ligar. Assim sendo, garante-se que as permissões de um utilizador se cingem a uma dada sala, tornando-se impossível que o utilizador X afecto à Sala A, interaja com a Sala B. Todas as restantes questões de segurança e controlo de acessos saem fora do âmbito desta dissertação e serão geridas por um componente ainda em desenvolvimento.

Como referido, uma forma de oferecer à aplicação cliente as funcionalidades colaborativas mencionadas foram então criados Client Widgets que funcionam como pequenas "aplicações" (do lado cliente) que podem ser incorporadas na aplicação principal e cujo intuito é disponibilizar a funcionalidade associada ao recurso sem que

quem desenvolve a aplicação principal necessite conhece-lo. Assim, garante-se a desejada abstracção para o nível aplicacional.

De entre as funcionalidades já referidas é possível dividi-las em duas "categorias"; as que, para o seu correcto funcionamento, necessitam de fazer uso do servidor flash (Red5) e as que não precisam. No primeiro grupo, denominado *Stream Widgets*, encontram-se as funcionalidades de *whiteboard*, *slideshow*, audio e video. No segundo, os *Media Widgets*, as de listagem de ficheiros, *upload* e *download*. De modo a garantir a maior ubiquidade possível no que toca a dispositivos e sistema operativos, optou-se por considerar a divisão nestas duas categorias de componentes e assim poder usar diferentes tecnologias e linguagens de programação que respeitem o *trade off* funcionalidade e universalidade.

Os widgets que abaixo abordarei partilham da característica de estabelecerem uma ligação bidireccional com o servidor de flash Red5 e utilizarem, para tal, o protocolo rtmp. Será ainda feito uso da tecnologia Flex pelas razões já apresentadas na secção 3.8.

Como anteriormente referido, foi necessário informar o controlador do recurso OpenMeetings sobre a actividade que passava directamente entre o widget e o servidor. Deste modo, serão indicados quais os eventos envolvidos em cada uma das actividades que de seguida serão abordadas. Referir-se-ão também quais os requisitos funcionais exigidos a cada um dos widgets. A ligação entre o recurso e o controlador é feita por RMI.

Considere-se a seguinte arquitectura apresentada que descreve a interacção entre os *Stream Widgets* e entre estes e o OpenMeetings:

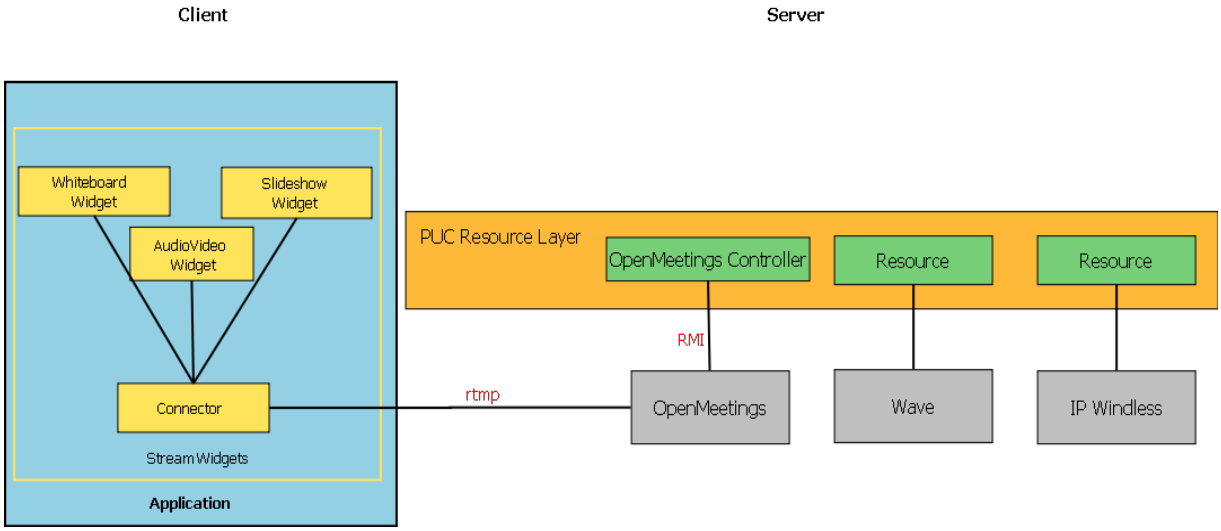

Figura 4: Representação da interacção dos widgets de Flash com o servidor

OpenMeetingsController – cada recurso tem um controlador associado. No caso do OpenMeetings, o respectivo controlador é o *OpenMeetingsController*. A sua função é receber os eventos enviados pelo recurso e implementar todo a lógica necessária consoante o tipo de evento recebido. Por exemplo, quando é recebido o evento a assinalar que um utilizador se ligou, são despoletadas as operações que fazem alterar o estado do utilizador na base de dados.

Connector – tem por objectivo gerir a ligação dos widgets de *whiteboard*, *slideshow* e audio e video ao servidor. Deste modo, mesmo que as três funcionalidades estejam a decorrer em simultâneo, existe apenas uma ligação ao servidor. Este componentes torna-se obrigatório visto que para o Red5, a cada ligação estabelecida corresponde um utilizador diferente. Sem o Connector, cada widget ligar-se-ia ao Red5 e, no pior dos casos, estabeleceria 3 ligações (são 3 widgets) que corresponderiam a 3 clientes diferentes, quando na verdade era apenas 1. Assim, sempre que nos referimos a uma comunicação directa entre o widget e o OpenMeetings, dever-se-à entender que a

mesma é divida em dois passos; primeiro entre o widget e o Connector e de seguida entre este e o recurso. E no sentido inverso, no caso de uma comunicação entre OpenMeetings e widget.

AudioVideo Widget – com este widget pretendeu-se tornar possível visualizar vídeo e áudio publicados por outros utilizadores. Para que tal aconteça é necessário que quem pretende partilhar a sua imagem e som os publique no servidor utilizado para o efeito - o Red5, integrado no OpenMeetings – e inicie, deste modo, uma conexão que permite enviar uma *stream* contendo o vídeo e o áudio capturados pela webcam e microfone, respectivamente. Já quem pretende visualizar, estabelece, igualmente, uma ligação com o servidor, de modo a poder receber o fluxo de dados (áudio e vídeo) que está a ser publicado pelo emissor.

Nesta funcionalidade existem 4 eventos que são enviados do OpenMeetings para a camada superior. São eles *Audio Start*/*Video Start* e *Audio Stop*/*Video Stop*. Estes, são enviados, respectivamente, quando se inicia a transmissão de áudio e vídeo e quando se cessa a mesma.

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- Publicar vídeo e áudio
- Parar a publicação de vídeo e áudio
- Visualizar, em qualquer altura, o vídeo e áudio publicado por outro

Whiteboard Widget – este módulo tem por objectivo oferecer a funcionalidade de whiteboard. É dada a possibilidade ao utilizador de escolher a ferramenta que pretende usar para desenhar, seja linha, elipse, rectângulo, desenho livre e escrita, bem como uma paleta de cores. O widget detecta o que o utilizador desenhou e envia uma mensagem contendo essa informação para o servidor. Este, por sua vez, difunde a mesma pelos restantes utilizadores ligados àquele *whiteboard*. Por último, o widget daqueles que recebem a mensagem, processa-a e apresenta o desenho sem que para tal haja necessidade de intervenção da pessoa.

Neste caso, o controlador do OpenMeetings recebe eventos apenas quando o *whiteboard* é iniciado e parado.

No que toca aos requisitos funcionais, este widget deve permitir o seguinte:

- Escolher a ferramenta a usar entre as seguintes: linha, elipse, rectângulo, desenho livre e escrita
- Escolher a cor da ferramenta
- Limpar o *whiteboard*
- Guardar o histórico das operações realizadas

Slideshow Widget – com a criação deste componente pretendeu-se tornar possível a visualização colaborativa de diapositivos. Deste modo, é possível ao apresentador iniciar uma apresentação para outros utilizadores e controlar a apresentação dos mesmos. Uma vez feita uma mudança de diapositivo por parte do apresentador, a mesma reflecte-se em todos os participantes. Internamente, o widget detecta para qual diapositivo foi feita a mudança e qual a apresentação em causa. Essa informação é enviada para o servidor que a difunde por todos quantos estejam a ver a mesma apresentação. Do lado de quem visualiza, o widget recebe a notificação e procede à necessária alteração de modo a apresentar o diapositivo correcto.

Na funcionalidade de *slideshow*, o recurso notifica o controlador quando é iniciada ou parada uma apresentação e quando há uma mudança de diapositivo. Deste modo, é sempre possível notificar outros clientes mesmo que estes não se encontrem ligados ao mesmo recurso (neste caso, o OpenMeetings).

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- O apresentador deve ter acesso à pré-visualização de todos os diapositivos e poder navegar por eles sem ser numa ordem sequencial
- Deve ainda ser possível navegar para o *slide* seguinte ou anterior sem recurso à pré-visualização

Ao contrário dos widgets acima descritos, a categoria dos que agora passamos a descrever não necessita de uma ligação permanente ao servidor nem de utilizar o Red5. Todas as operações são realizadas fazendo uso de *servlets*. Uma *servlet* é uma classe Java, existente do lado do servidor e que interage com os clientes, utilizando o modelo *request/response*. Normalmente, as *servlets* utilizam o protocolo HTTP. Assim, não havendo necessidade de utilizar o servidor flash optou-se por desenvolver estes widgets em HTML e JavaScript, fazendo uso do GWT, e, deste modo, torná-los mais portáveis e mais facilmente compatíveis com diferentes dispositivos.

Mais uma vez, também aqui existe interacção entre o recurso (OpenMeetings) e o respectivo controlador (OpenMeetingsController). A arquitectura destes widgets é a seguinte:

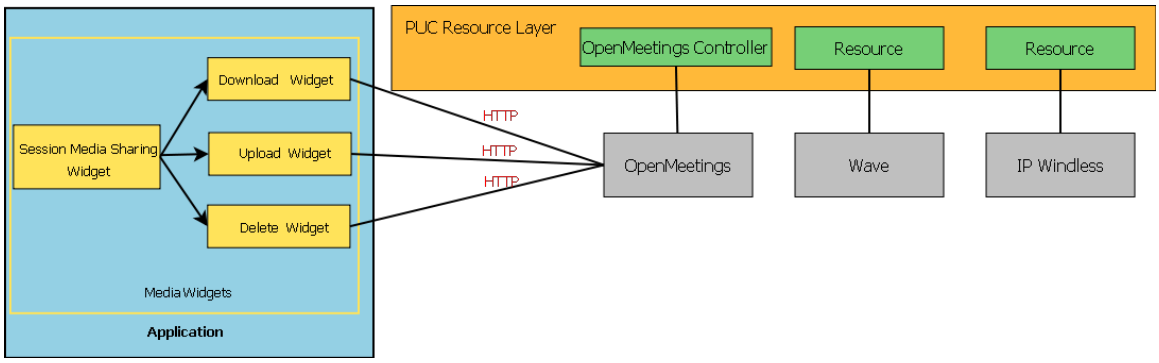

Figura 5: Representação da interacção dos widgets de GWT (JavaScript) com o servidor

Session Media Sharing Widget – este widget tem por objectivo listar e disponibilizar para *download* os ficheiros partilhados numa sessão; é feito um pedido ao Resource Manager que disponibilizará a referida lista. Para além disto pretendeu-se ser possível enviar ficheiros para o servidor, de modo a ficarem disponíveis na sessão. Por último é ainda disponibilizada a opção de remover um ficheiro constante de uma sessão. Os processos de *download, upload* e remoção são concretizados por outros três widgets independentes (apresentados mais à frente), chamados pelo Session Media Sharing.

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- Apresentar uma lista com os ficheiros partilhados na sessão

- Disponibilizar a possibilidade de *download* de um ficheiro por invocação do Download Widget
- Apagar um ficheiro por invocação do Delete Widget
- Oferecer a funcionalidade de *upload* por invocação do Upload Widget

Download Widget – este componente tem como função facilitar a descarga de um ficheiro cujo nome e localização lhe é passado. Uma vez recebidos esses dados é gerado um URL que permite o *download* do ficheiro. A partir do momento em que o utilizador pressiona o *link* para aceder ao referido URL, são utilizadas as capacidades internas do *browser* para fazer um pedido (GET) a uma *servlet* que se encarrega de lhe enviar o ficheiro.

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- Facilitar o processo de descarga através da criação de um URL
- Exibir janela de confirmação antes do pedido ser efectuado à *servlet*

Upload Widget – este é o widget responsável por enviar o ficheiro que se pretende disponibilizar para partilha. O processo é efeito recorrendo a uma *servlet* e utilizando o método POST. Aquando da sua correcta gravação em disco, o OpenMeetings notifica o seu controlador que um novo ficheiro foi adicionado. Mais, se o ficheiro for uma apresentação (um .ppt, por exemplo) é ainda enviado outro evento indicando a existência de um novo *slideshow*.

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- Navegar em disco local de modo a encontrar o ficheiro que se pretende enviar
- Exibir janela de confirmação de sucesso ou falha da transferência

DeleteFile Widget – este módulo pretende oferecer a possibilidade de um ficheiro ser removido. O widget comunicará com uma *servlet* do OpenMeetings, fazendo um pedido GET, indicando qual o ficheiro que deve ser apagado. O servidor encarregar-se-à de o apagar e notificar a camada superior sobre o sucedido.

Ao nível de requisitos funcionais, este widget deve permitir o seguinte:

- Exibir janela de confirmação antes de proceder, efectivamente, ao pedido de remoção
- Exibir janela de confirmação do sucesso ou falha

5. Implementação

Analisado o desenho da solução, na secção anterior, nesta serão abordados os detalhes de implementação da referida arquitectura. Será feito um *zoom in* a cada módulo e explicar-se-à o modo como foi implementado.

Uma vez que houve desenvolvimento quer do lado do cliente, quer do servidor, optou-se por dividir as seguintes secções nestas duas componentes.

5.1. Cliente

5.1.1. Suporte à execução

Um dos requisitos fundamental para que os widgets de flash (Stream Widgets) funcionem é que o dispositivo tenha o plugin de flash instalado. Se nos computadores isso não é problema[29], nos dispositivos móveis, por exemplo, já poderá ser. Alguns sistemas operativos de dispositivos móveis já trazem o plugin de flash instalado, como é o caso do Android 2.2, outros trazem uma versão Lite do mesmo, como acontece com a versão 2.1 do referido sistema. Neste último cenário o flash apenas funciona dentro do browser, não sendo possível utilizá-lo fora deste.

Já no que concerne aos Media Widgets, os requisitos passam pela existência de um *browser engine* no terminal do cliente. O *browser engine*, muitas vezes generalizado apenas para *browser* é um componente que interpreta várias linguagens (HTML, XML, etc.) e informações de formatação (CSS, XLS, entre outras). Mais uma vez, também aqui, é possível fazer uma distinção entre os comuns computadores e os terminais móveis. Se

nos primeiros, os *browsers* mais comuns apresentam boa capacidade de resposta para os requisitos necessários aos widgets, já nos segundos, a situação não é a mesma. Mais à frente, na secção de Testes e Avaliação, é possível analisar como estas diferenças influenciam o comportamento dos widgets, no que toca às funcionalidades por eles oferecidas.

5.1.2. Stream Widgets

Os Stream Widgets são o conjunto dos widgets que se baseiam da tecnologia flash e oferecem as seguintes funcionalidades: video-conferência, *slideshow* e *whiteboard*. Em comum partilham também do facto de comunicarem com o OpenMeetings através do Red5, utilizando o protocolo rtmp.

Tendo por base a *Figura 4* analisemos, de seguida, cada módulo existente do lado do cliente.

Connector

A aplicação pode fazer uso de várias funcionalidades em simultâneo, tais como *whiteboard* e publicação de vídeo e audio. Cada uma destes recursos oferecidos necessita de ligação ao OpenMeetings, nomeadamente ao servidor de flash para poder enviar e receber dados em tempo real. Sem a utilização do Connector, cada módulo, de cada funcionalidade, teria que estabelecer uma ligação ao servidor. Esta abordagem cria, à partida, dois problemas: um é o consumo desnecessário de recursos quer do cliente, quer do servidor, de modo a gerir as ligações que, como anteriormente foi referido, são persistentes; o outro pretende-se com o facto já mencionado de, para o OpenMeetings, cada ligação estabelecida dizer respeito a um utilizador diferente, o que, no exemplo referido, significaria que o servidor iria reconhecer como dois utilizadores aquele que, no fundo, é apenas um.

De modo a ultrapassar o problema evidenciado, criou-se o módulo Connector. Este estabelece a única ligação ao servidor e trata de enviar as mensagens para o mesmo e recebe-las, identificando quais os widgets a que se destinam. Fazendo uma analogia com o tráfego numa rede doméstica, este módulo poder-se-à comparar a um *router* e os restantes widgets aos computadores da rede.

Para que o Connector se ligue ao servidor necessita da sala à qual o utilizador se pretende juntar, bem como o identificador único do participante, doravante designado por *sid* (session ID). Com estes dados é então estabelecida uma ligação utilizando o protocolo rtmp e são invocados métodos do lado do OpenMeetings de modo a marcar o utilizador como "*logged in*" e a registá-lo convenientemente no sistema. Eis algumas linhas de código que ilustram este processo:

```
(1) nc=new NetConnection();
(2) nc.client=this;
(3) nc.objectEncoding=ObjectEncoding.AMF3;
(4) nc.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
(5) nc.connect("rtmp://" + ip + ":1935/openmeetings/" + roomid);
```

Na primeira linha é criado um objecto do tipo NetConnection que representa a ligação ao servidor. Na segunda linha é indicado o objecto sobre o qual as *callbacks* do servidor devem ser enviadas. No ponto (3) é definida a codificação dos dados, neste caso é AMF3. A explicação do sobre a codificação dos dados é feita no secção de Tecnologias Relacionadas (3.9). No ponto seguinte, é adicionado um *event listener* que tratará de escutar os eventos do tipo NET_STATUS. É deste modo que sabemos se a ligação foi estabelecida com sucesso ou houve falhas e quando a mesma é interrompida. Por último, o ponto (5) representa o exacto momento em que se tenta conectar a uma dada sala do servidor que se encontra disponível num determinado IP.

Uma vez efectivada a ligação é utilizado o método *call*, que permite invocar métodos existentes do lado do servidor de flash, com o objectivo de marcar o utilizador como "*logged in*" e registá-lo no sistema. Eis as duas linhas de código utilizadas para tal:

```
(1) nc.call("xmlcrm.markSessionAsLogedIn", null, sid);
(2) nc.call("xmlcrm.loginUserByRemote", null, sid);
```

Deste modo, são invocados dois métodos no servidor chamados "markSessionAsLogedIn" e "loginUserByRemote", aos quais é passado identificador do utilizador e que contém o código necessário para fazer "*log in*" do utilizador no sistema.

AudioVideoWidget

Este widget é utilizado quando se pretende publicar ou visualizar o áudio e o vídeo. Aquando do "*log in*" no OpenMeetings, este atribui a cada utilizador um *broadcastID* único. Este é o atributo que irá ser utilizado para a publicação e visualização. Ou seja, a pessoa que publica o áudio e vídeo, fá-lo utilizando esse valor; quem pretende receber o que outro está a publicar, utiliza o *broadcastID* dessa mesma pessoa para assim ter acesso à respectiva *stream*.

A publicação do áudio e vídeo exige a criação de uma *stream*, com um só sentido, pela qual os dados possam navegar entre a aplicação flash e o servidor (Red5). A esta *stream* a Adobe designou-a por NetStream. Uma vez criada é anexado o áudio e o vídeo e por último é publicada. Ou seja, é enviada para o servidor a *stream* capturada. Eis algumas linhas de código que permitem realizar o processo descrito:

```
(1) nsPub=new NetStream(nc);
(2) nsPub.attachCamera(Camera.getCamera());
(3) nsPub.attachAudio(Microphone.getMicrophone());
(4) nsPub.publish(myBroadcastID);
```

Na primeira linha é criado o objecto NetStream que funcionará dentro da NetConnection anteriormente criada. As segunda e terceira linhas anexam a fonte de vídeo e áudio à *stream*. Por último, a quarta linha publica para o servidor o áudio e vídeo capturados com o *broadcastID* correspondente ao utilizador que se prepara para iniciar a partilha.

Já para quem pretende visualizar o processo é ligeiramente diferente. Neste caso, para além da instanciação do objecto NetStream, é necessário criar um objecto do tipo

Video e anexar essa NetStream. Este objecto Video permite reproduzir vídeos FLV gravados localmente ou num servidor, bem como transmissões em directo, como é o caso. As linhas de código que se seguem ilustram o processo explicado.

```
(1) nsCli=new NetStream(nc)
(2) var vid:Video=new Video();
(3) vid.height=height;
(4) vid.width=width;
(5) vid.attachNetStream(nsCli);
(6) nsCli.play(othersBroadcastID);
```

Tal como quando se pretende publicar áudio e vídeo, também aqui se cria primeiramente uma NetStream. De seguida cria-se o objecto Video, especificando as dimensões, assim como a "fonte" dos dados (NetStream). Por último faz-se *play* dando como argumento o *broadcastID* de quem pretendemos visualizar. O servidor ao receber a notificação de *play* inicia a transmissão da *stream* (áudio e vídeo) do cliente que já está a publicar. A partir deste momento, o servidor está a receber a *stream* capturada pelo cliente 1 e a retransmiti-la para o cliente 2, que pretende visualizar.

Slideshow Widget

Este widget foi desenhado de modo a separar a lógica (SlideshowCore) dos controlos (SlideshowController). Assim, um dos componentes faz o processamento dos diapositivos (como a actualização do ecrã, etc) e outro contém os controlos para a navegação entre *slides*.

Esquematicamente o widget apresenta-se da seguinte maneira:

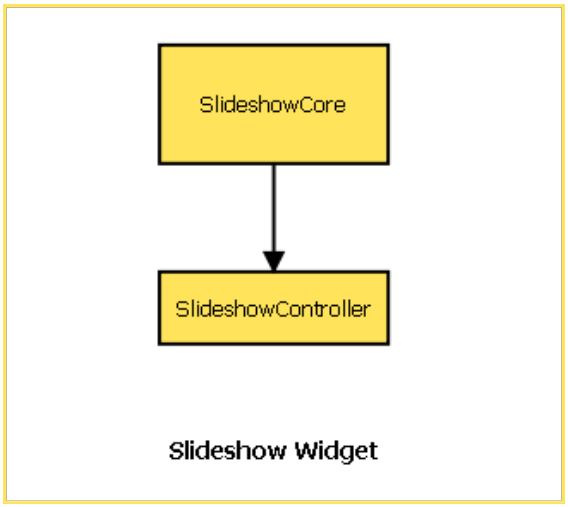

Figura 6: Detalhe do Slideshow Widget

Foi também necessário definir um padrão no que toca aos eventos, de modo a precisar qual o formato e o conteúdo da mensagem a enviar quando existe uma transição de *slides*. No processo de escolha do formato teve-se em consideração a conversação de dados entre Java e ActionScript[63] e, deste modo, optou-se pela utilização de um vector (String). Feita esta opção definiram-se os parâmetros necessários a enviar. Eis a escolha tomada:

```
[fileName, parentPath, roomID, slideNumber]
```

FileName indica o nome do *slideshow* que está a ser mostrado; parentPath a directoria onde se encontra o ficheiro; roomID a sala onde decorre a apresentação; slideNumber o número do *slide* em uso.

Quando um utilizador faz o envio de uma apresentação, por exemplo um ficheiro .ppt, para o servidor, este, internamente, converte-o num ficheiro swf. Swf é um formato de arquivo desenvolvido pela Adobe e que, neste caso, conterá todos os diapositivos. Também, para que seja possível outros dispositivos que não suportam flash visualizarem a apresentação (como no já referido caso da *set-top box*), alterou-se o

processo de conversão inicial e acrescentou-se a geração de imagens (jpeg), uma por cada *slide*. No processo inicial, o servidor já criava *thumbnails* para cada diapositivo, de modo a permitir a pré-visualização de todos os *slides* e facilitar a navegação. Assim, e uma vez que as ferramentas necessárias à geração de imagens já se encontravam disponíveis, tornou-se simples alcançar o pretendido.

SlideshowCore

Como referido anteriormente este módulo contém toda a lógica associada a uma apresentação. Isto é, uma vez recebida a mensagem acima descrita, enviada pelo Connector, esta é processada e através dos campos *fileName, parentPath* e *roomID* é construído o URL que será usado para descarregar o ficheiro swf. De modo a tornar o swf navegável este é carregado dentro de um objecto MovieClip que contém uma *timeline*. Assim, é possível navegar por *frames*, onde cada uma corresponde a um *slide*. Realizado este processo de carregamento da apresentação, a mesma começa sempre por mostrar o primeiro *slide*. É nesta altura que é enviada a notificação para o OpenMeetings notificando que se iniciou uma apresentação, no *slide* número 1. O servidor, ao receber a mensagem dispara um evento para o OpenMeetingsController indicando que foi iniciada uma apresentação numa determinada sala. Nesse evento é referido qual o URL da imagem correspondente ao *slide* que se encontra em apresentação. Este evento poderá servir para notificar outros terminais não ligados ao Red5, como o exemplo já referido da *set-top box*. Para além disso, o próprio OpenMeetings difunde por todos os outros participantes da sala ligados através do Red5, os eventos recebidos. O Connector, do lado de quem visualiza o *slideshow*, recebe a mensagem e encaminha-a para o SlideshowCore que tratará de iniciar a apresentação no *slide* indicado.

SlideshowController

A função deste componente é disponibilizar os botões de controlo para a navegação no *slideshow*. De momento os existentes são dois e permitem avançar ou retroceder no *slide*. Uma vez pressionado o botão, este componente detecta o "click" e chama a função *nextSlide* ou *previousSlide* do SlideshowCore, conforme a situação aplicável. Uma vez recebido esse pedido, o Core procede à mudança de *slide* e envia a notificação para o OpenMeetings que fará o que acima foi descrito. Eis o fluxo, a vermelho, da mensagem desde o pedido de mudança de *slide*, por parte do apresentador, até à recepção do lado de quem visualiza:

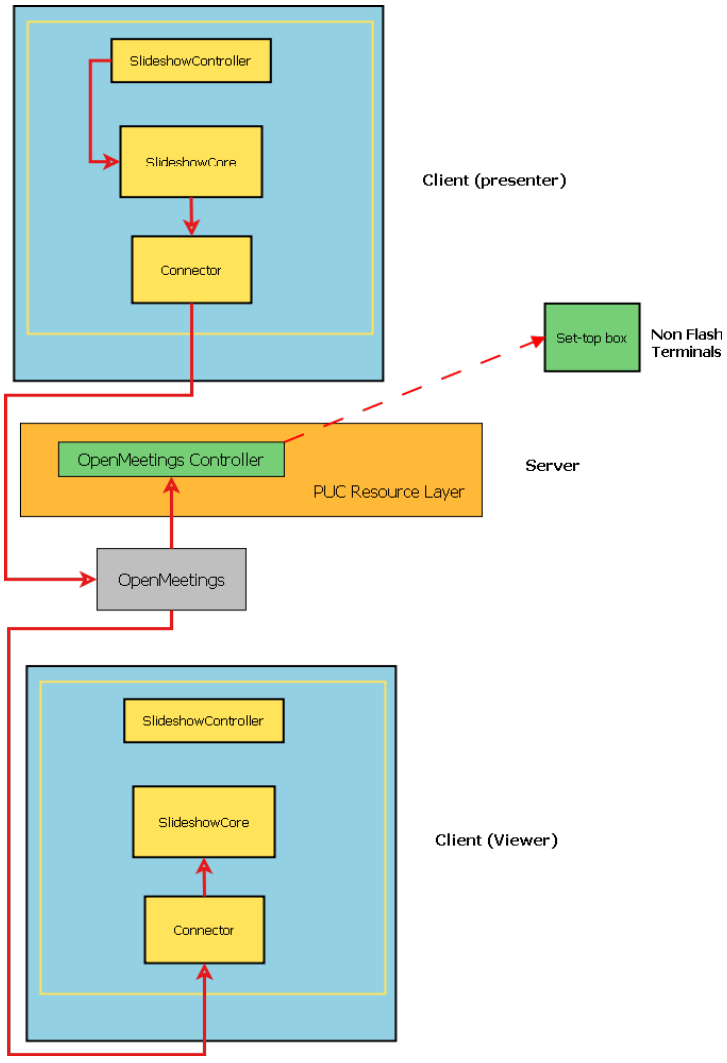

Figura 7: Fluxo de mensagem despoletada pela mudança de slide

Whiteboard Widget

Este widget permite a realização de desenhos, num quadro virtual, utilizando diferentes ferramentas e cores. Tal como no widget anterior, pretendeu-se separar a lógica dos desenhos (WhiteboardCore) daquilo que se considera mais "acessório", como as cores (WhiteboardColorPicker). Eis o esboço da arquitectura de alto nível:

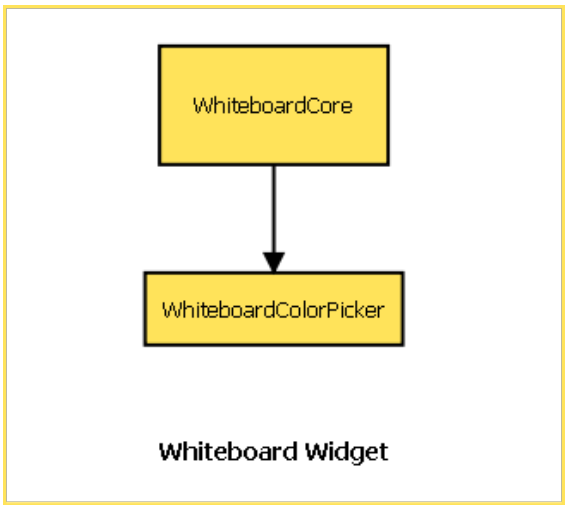

Figura 8: Detalhe do Whiteboard Widget

Assim, o WhiteboardCore representa a parte lógica, contendo o processamento dos desenhos. Isto é, é este módulo que tem como função "desenhar" os elementos presentes no *whiteboard* (rectas, círculos, etc.). O WhiteboardColorPicker fornece a paleta de cores que o utilizador pode escolher.

À semelhança do Slideshow, também aqui foi necessário definir quais os parâmetros enviados nas mensagens. Os mais importantes são uma estampilha temporal que poderá vir a ser usada para posterior sincronização (funcionalidade ainda não implementada), a ferramenta usada no desenho e o ponto inicial e final. A função de sincronização poderá vir a servir para o caso em que um utilizador se junta à sala posteriormente e, deste modo, ser possível receber os desenhos já efectuados até ao instante em que se juntou. Quando a ferramenta utilizada é a de *handwrite* (escrita livre),

para além dos parâmetro já referidos, é também enviado um vector contendo os diversos pontos que formam os pequenos segmentos de recta que representam a escrita.

WhiteboardCore

Quando o *whiteboard* é criado, o *core* prepara uma área branca, nas coordenadas e tamanho indicados pela aplicação (cliente) principal, onde fica à escuta de eventos do rato. Para além disso, disponibiliza as ferramentas para desenho e pede ao WhiteboardColorPicker para criar uma paleta de cores. Por defeito, a ferramenta seleccionada é a linha e a cor é preta. Uma vez instanciado o *whiteboard* este passa a detectar os cliques do rato e a reproduzir o que é desenhado conforme a ferramenta seleccionada. Foi necessário garantir que o utilizar, ao pressionar o rato, via o seu desenho aparecer em tempo real e não apenas quando o largasse. Para que tal acontecesse foi necessário que após o primeiro clique, todo o movimento fosse monitorizado e o desenho sendo actualizado até o botão do rato deixar de ser pressionado. Eis algumas linhas de código que ilustram o desenho de uma recta:

```
(1) case "line":
(2) whiteboardComponent.graphics.lineStyle(2, this.selectedColor, … );
(3) whiteboardComponent.graphics.moveTo(this.initX, this.initY);
(4) whiteboardComponent.graphics.lineTo(e.localX, e.localY);
(5) e.updateAfterEvent();
(6) break;
```

No caso da ferramenta seleccionada ser a linha (1), é definido o estilo do traço com a cor escolhida (2) e desenha-se a linha do ponto inicialmente clicado (3) até à posição actual do rato (4). A linha (5) permite estar constantemente a ler a posição do rato e assim ir actualizando o desenho da linha em tempo real. Uma vez solto o botão do rato é enviada uma mensagem para o OpenMeetings indicando o que foi desenhado, de modo a poder ser reproduzido no *whiteboard* dos restantes participantes. Recebida essa mensagem, pelo Connector, este reencaminha-a para o WhiteboardCore que a analisa e, consoante a ferramenta a usar, chama o método indicado para reproduzir o desenho.

Ao contrário do que acontece com o *slideshow,* o OpenMeetings não envia uma notificação para o seu controlador sempre que há uma mudança. Neste caso, existem apenas dois eventos disparados para a camada superior: o de início do *whiteboard* e o de fim. Tal deve-se ao facto de todos as aplicações que pretendam utilizar esta funcionalidade necessitarem ligar-se directamente ao OpenMeetings. Deste modo, recebem sempre a notificação enviada pelo próprio servidor. O comportamento difere do *slideshow* porque, como já foi referido, no caso da apresentação, pretende-se que todos os clientes, mesmo que não estejam ligados ao Red5 possam vê-la.

WhiteboardColorPicker

Este componente tem por objectivo criar e disponibilizar uma paleta de cores a usar nos desenhos. É ainda responsável por detectar qual a cor escolhida e, a pedido do WhiteboardCore, indicar a cor seleccionada.

Snapshot

Uma das funcionalidades também disponibilizadas foi a de *snapshot* ao *whiteboard*. Com isto pretende-se que, a pedido da plataforma (PUC), seja registado o estado do *whiteboard* naquele dado momento e o mesmo gravado como imagem. Para tal, é criado um Bitmap que servirá para criar a referida imagem no formato JPEG[64]. O envio para o servidor é feito por HTTP (método POST) através de uma *servlet* que recebe os dados em base64. Uma vez que esta funcionalidade não estava prevista no OpenMeetings, foi necessário fazer modificações do lado do servidor, nomeadamente, a criação da *servlet* usada para efeito.

Um dos problemas surgidos foi o facto de os dados serem enviados num formato binário criado dinamicamente e, como tal, o "ficheiro" não conter um elemento de fronteira que naturalmente existe quando é submetido um ficheiro fazendo *browse* pelo

disco local (o tradicional método de *upload*). Isto é, criando dinamicamente os dados a enviar, torna-se complicado construir um ficheiro onde são especificados os *headers* e os dados relativos ao corpo e assim definir, claramente, o elemento de fronteira entre as duas partes. Deste modo, não é possível enviar os dados (imagem) e ao mesmo tempo passar parâmetros (através do URL) uma vez que *servlet* não consegue determinar a fronteira que separa os dados da imagem dos dados relativos aos parâmetros. Como forma de ultrapassar este problema, quando se enviam os dados para a *servlet* são, na realidade, feitos dois POST. O primeiro, contém os parâmetros necessários para que a *servlet* saiba que dados vão ser enviados de seguida e qual a acção a tomar. No segundo são apenas enviados os dados que formam a imagem. Uma vez que os pedidos são sequenciais é preciso que, aquando do primeiro, do lado da *servlet* (servidor) os parâmetros sejam guardados em variáveis de sessão, válidas durante um determinado período de tempo. Assim, quando a *servlet* receber o segundo POST pode ler o valor dos parâmetros já guardados nas referidas variáveis.

O código que se segue pretende ilustrar a criação da imagem, do lado do cliente:

```
var jpgSource:BitmapData=new BitmapData(window.width, window.height);
jpgSource.draw(window);
var jpgEncoder:JPGEncoder=new JPGEncoder(70);
jpgStream=jpgEncoder.encode(jpgSource);
this.doPost(jpgStream, this.firstTimePosting);
```

Uma vez criada a imagem é então feito o "duplo" POST como se pretende ilustrar no código presente no Anexo em 9.1..

5.1.3. Media Widgets

Os Media Widgets, representados na Figura 5, são responsáveis por oferecer as funcionalidades de listagem de ficheiros da sessão, *download* e *upload*. Ao contrário dos widgets atrás apresentados, estes não necessitam fazer uso de flash. Deste modo, não há necessidade de estabelecer a referida ligação permanente ao servidor de flash (Red5), que atrás se viu ser necessário. Assim, nos widgets que seguidamente serão abordados, o

recurso OpenMeetings comporta-se como um servidor web que disponibiliza documentos e envia notificações para o seu controlador. Uma outra consequência da não existência da ligação permanente ao servidor é que as acções realizadas através destes widgets apenas despoletam o envio de notificações para o OpenMeetingsController, ao contrário do que antes sucedia (onde os participantes ligados à mesma sala recebiam os eventos directamente através do servidor).

De referir ainda que estes widgets são integrados e controlados por outros, em desenvolvimento no âmbito de outra dissertação[21], que manipulam os dados da sessão, como a lista de participantes, os seus contactos, etc. Estes, quando necessário, invocam o aparecimento do SessionMediaSharing Widget que, por sua vez, poderá invocar os Download, Upload e Delete Widgets, para obter as funcionalidades pretendidas.

SessionMediaSharing Widget

A principal função deste widget é disponibilizar ao utilizador uma listagem do ficheiros partilhados numa sessão. Simultaneamente, incorpora outros três widgets, oferecendo outras tantas funcionalidades: descarregar o ficheiro, apagá-lo e partilhar um novo através de *upload*.

Todos os ficheiros partilhados numa sessão estão registados na base de dados da plataforma (PUC), sob a forma de *MediaUAData. MediaUAData* corresponde a uma classe que contém informação sobre um determinado ficheiro. Dessa informação constam o nome do ficheiro, o URL através do qual pode ser acedido e se se encontra partilhado na sessão (isto é, se está ou não acessível). Assim, este widget, está preparado para, ao ser instanciado, receber um vector contendo *MediaUAData* que tratará de listar, numa tabela, conforme o ficheiro se encontre ou não partilhado. Uma vez que se fez uso do GWT (programação em Java) mas os objectos *MediaUAData* recebidos são em JavaScript, foi necessário mapea-los para poderem ser manipulados. Este tipo de mapeamento consiste, essencialmente, em criar *getters* para acedermos aos campos do objecto, como a seguir é exemplificado:

```
public class MediaUAData extends JavaScriptObject {
	protected MediaUAData(){}
	public final native int getID() /*-{ return this.id; }-*/;
	public final native String getResourceElementName() /*-{
		return this.resourceElementName; }-*/;
	public final native String getResourceElementAddress() /*-{
		return this.resourceElementAddress; }-*/;
(…)
}
```

Sendo a linguagem de programação usada em GWT Java, para podermos escrever JavaScript nativo é necessário utilizar a anotação "/*-{ }-*/" como em cima é ilustrado.

Feita a listagem dos ficheiros é ainda necessário oferecer a possibilidade de *download* dos mesmos, bem como oferecer a possibilidade de remoção. Assim, sempre que um utilizador clique num dos ficheiros listados, é adicionado o Download Widget que, mostrando uma janela de confirmação, desencadeia o *download* do mesmo. A remoção do ficheiro faz-se clicando numa cruz, que surge à frente do nome, e que faz despoletar a execução do Delete Widget.

Este widget está ainda preparado para que a lista de ficheiros seja actualizada a pedido. Isto é, é possível que outros widgets interajam com este e actualizem a lista. Para tal foram criadas duas funções de adição e remoção que podem ser chamadas em qualquer altura e forçam a alterações na tabela de ficheiros disponíveis.

A figura que se segue mostra o SessionMediaSharing Widget, juntamente com os Upload e Delete Widgets:

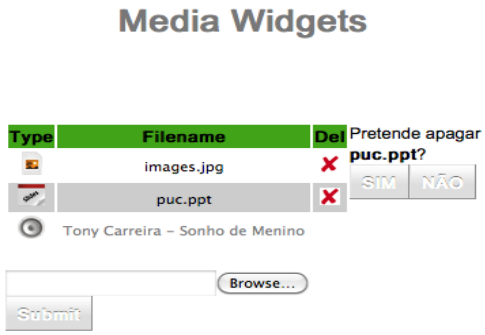

Figura 9: SessionMediaSharing Widget juntamente com Upload Widget e Delete Widget

Download Widget

Este componente é normalmente invocado pelo SessionMediaSharing Widget e recebe como parâmetros o URL e o nome do ficheiro a descarregar. A informação quando é recebida, é processada de modo a ser criado um *link* e surgir uma janela de confirmação ao utilizador. Este pode optar por cancelar a operação ou prosseguir. No caso de optar pela segunda hipótese é desencadeado o processo normal de *download* através das funcionalidades nativas do *browser*. Deste modo, é feito um pedido a uma *servlet*, integrada de origem no OpenMeetings, chamada DownloadHandler, que se encarrega de verificar se o ficheiro existe e, em caso afirmativo, devolve-o. Concluído o *download*, o widget termina. O mesmo acontece caso a opção tomada seja a de cancelar o processo.

Upload Widget

O widget de *upload* surge também, normalmente, associado ao widget SessionMediaSharing, aparecendo por baixo da lista de ficheiros partilhados na sessão. Este widget de *upload*, consiste, essencialmente, num formulário que permite localizar no disco local o ficheiro pretendido e num botão de envio (*submit*). Uma vez pressionado este último, o ficheiro é enviado para o servidor que o guarda no seu disco e envia uma notificação ao controlador indicando que existe um novo ficheiro associado à sessão.

Existe ainda a possibilidade de o ficheiro enviado ser convertido. Por exemplo, caso se trate de um *powerpoint* (ou similar) ou de um pdf, é criada uma apresentação a partir do mesmo, ou seja, é criado o ficheiro swf, bem como as imagens correspondentes a cada *slide*. Terminada a conversão é disparado um novo evento para o OpenMeetingsController sugerindo a existência de uma nova apresentação.

Delete Widget

Como o nome sugere, este widget oferece a possibilidade de remoção de um ficheiro. O utilizador, ao clicar na cruz que surge em SessionMediaSharing faz despoletar a criação deste widget que recebe como argumento o *resourceElementAddress* (endereço de *download*) constante em MediaUAData. Mais uma vez, também aqui, surge uma janela de confirmação ao utilizador. Feita essa confirmação, são lidos, do endereço recebido, os parâmetros necessários. Visto que o OpenMeetings não suporta este tipo de operações, fazendo uso de *servlets* (de origem é apenas permitido invocando um método, através da ligação do Red5), foi necessário criar uma nova que recebendo pedidos de remoção, procedesse à eliminação do referido ficheiro no servidor. Os parâmetros necessários à *servlet* serão abordados na secção destinada a esta. Uma vez removido com sucesso o ficheiro, é enviada uma notificação para o controlador do recurso, dando conta do facto ocorrido.

5.2. Servidor

O OpenMeetings é uma plataforma que tem como objectivo dar suporte a serviços de videoconferência, visualização colaborativa de diapositivos e *whiteboarding,* etc. Como pontos de entrada para os clientes, dispõe de um servidor de Flash e de um conjunto de *servlets,* algumas já existentes e outras que houve necessidade de implementar que seguidamente serão abordados.

5.2.1. Red5

O Red5 é um servidor de Flash, de código aberto, escrito em Java[65]. É um dos componentes integrados no OpenMeetings e tem como função permitir ligações a este utilizando o protocolo de flash (rtmp). Apesar de, no âmbito da realização desta dissertação, não ter sofrido nenhuma alteração em relação ao seu estado inicial, trata-se

de um componente de grande importância uma vez que é com base neste que os Stream Widgets foram implementados, sendo assim essencial para o desenvolvimento das funcionalidades de áudio/vídeo, *whiteboard* e *slideshow*.

5.2.2. Servlet

Uma *servlet* baseia-se numa classe Java, que permite implementar um serviço Web do lado do servidor suportando novas funcionalidades. Nesta implementação estas *servlets* encontram-se alojados no servidor Jetty, sendo acessíveis aos clientes utilizando o modelo *request/response* sobre o protocolo HTTP.

As *servlets* que de seguida abordamos são o ponto de ligação entre os Media Widgets e o OpenMeetings. Ao contrário dos Stream Widgets que se ligavam ao servidor através do Red5, estes estabelecem a comunicação acedendo a *servlets*. Assim, para garantir a portabilidade dos Media Widgets, não se pretendeu ficar dependentes, para a sua implementação, do Flash e Red5. Optámos por adicionar ao OpenMeetings a possibilidade de serem feitos pedidos através de *servlets* e recorrendo a HTTP (GET e POST) para efectuar diversas operações sobre ficheiros. Analisemos, de seguida, as três principais *servlets* (ProcessExternalRequests, ProcessBinaryData e UploadHandler), no que toca às suas características.

1. ProcessExternalRequests

Uma vez que existia a necessidade de realizar a operação de *delete* sobre ficheiros mas esta não era, nativamente, uma funcionalidade suportada pelo servidor, optou-se por criar uma *servlet* que desse resposta a este requisito e que permitisse, com um simples pedido GET (HTTP), apagar um determinado ficheiro.

Eis a parte do código mais representativo da *servlet*:

```
protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException,IOException{
	String action = request.getParameter("action") ;
	String sid = request.getParameter("sid") ;
	String fileName = request.getParameter("fileName") ;
	String moduleName = request.getParameter("moduleName") ;
	String parentPath = request.getParameter("parentPath") ;
	long room_id = Long.parseLong(request.getParameter("room_id")) ;
	boolean success = false ;
	if (action.compareTo("delete") == 0)
		success = ConferenceLibrary.getInstance().deleteFile(sid,
		fileName, moduleName, parentPath, room_id) ;
	if (success){ // Send event to controller }
}
```

De modo a poder ser identificado o ficheiro, foram necessários definir alguns parâmetros a serem passados no pedido. O primeiro, "action", define a acção a ser realizada. De momento a *servlet* está apenas preparada para o caso da acção ser *delete*, no entanto, é possível, mais tarde, vir a suportar novas acções. O segundo parâmetro, "sid" (identificador único) torna possível a identificação do utilizador que fez o pedido. Os restantes, "filename", "moduleName", "parentPath" e "room_id", permitem eliminar o ficheiro, visto que utilizados em conjunto tornam possível construir o caminho do mesmo, no disco do servidor. Uma vez realizada, com sucesso, a operação é enviado um evento para o controlador do recurso indicando qual o ficheiro que foi eliminado.

2. ProcessBinaryData

Como referido anteriormente, foi necessário criar uma *servlet* que recebesse a imagem criada pela função de *snapshot* ao *whiteboard*. Foi também indicado que, na verdade, seriam feitos dois POSTS sequenciais: o primeiro contendo os parâmetros necessários e o segundo os bytes que formam a imagem. Como no segundo POST os parâmetros não são enviados foi preciso guarda-los aquando do envio do primeiro. Para tal, utilizam-se as variáveis de sessão, através uso do objecto HttpSession[66]. Uma vez

registados esses parâmetros, durante um determinado período de tempo, cada vez que à *servlet* chega um novo pedido esta irá ler os dados necessários, já guardados em HttpSession. Ultrapassado este processo, a *servlet* receberá o segundo POST contendo os bytes da imagem codificados em base64. Irá então descodificar os mesmos e proceder à escrita dos bytes em disco, no servidor. De modo a garantir a unicidade do nome das imagens e permitir, depois, ordenar temporalmente as *snapshots*, os ficheiros são guardados com uma estampilha temporal associada ao nome.

O código da *servlet* que ilustra o processo descrito, pode ser visto em Anexo, no ponto 9.2.

3. UploadHandler

UploadHandler é a *servlet* original do OpenMeetings, utilizada quando se pretende enviar ficheiros para o servidor. Uma vez que é preciso notificar o OpenMeetingsController aquando de um novo ficheiro recebido, houve a necessidade de intervir ao nível desta *servlet* e dotá-la do mecanismo necessário para o efeito. Existem dois possíveis eventos a ser despoletados no final do processo de *upload*. O primeiro acontece quando a *servlet* guarda o ficheiro e torna-se necessário notificar o controlador que este foi adicionado à sessão. O segundo surge apenas quando o ficheiro enviado se trata de uma apresentação e, nesse caso, depois da criação do ficheiro swf e respectivas imagens, é necessário informar o OpenMeetingsController da existência de uma nova apresentação na sessão.

5.2.3. Eventos

Um dos pontos importantes da solução pensada reside no envio de eventos do recurso para o seu controlador. Tal processo começou a ser desenvolvido no âmbito de outra dissertação[16] e toda a estrutura desenvolvida se manteve. Contudo, foi necessário

proceder a algumas alterações e colocar, explicitamente, o código de envio nos devidos locais do servidor.

Existem duas classes chave em todo este processo: a primeira é a que representa o evento e é designada por OMEvent; a segunda é uma enumeração que contém todos os tipos de eventos existentes e denomina-se OMEventType. Em qualquer uma delas foi necessário realizar alterações. No caso da enumeração houve uma renomeação de alguns eventos, remoção de outros e ainda a adição de novos. A classe representativa do evento (em Anexo, 9.3) foi também alterada pois existiu a necessidade de serem adicionados novos campos que fossem passados para o controlador.

Como referido, foi também necessário intervir (no servidor) de modo a colocar, nos locais apropriados, o código necessário para o envio dos eventos para o OpenMeetingsController. Nesses locais, é invocada a classe EventDispatcher que estabelece a conexão RMI ao controlador, constrói o evento e envia-o.

O código que ilustra a chamada da classe EventDispatcher é o seguinte:

```
	EventDispatcher.dispatchEvent(roomClient.getSid(), roomID,
		roomClient.getUserip(),   ScopeApplicationAdapter.getInstance(),
USER_IN);
```

Ao ser chamado o método que envia o evento é importante passar alguns parâmetros que identifiquem, com clareza, o mesmo. Assim, é necessário identificar o cliente que fez despoletar o evento, através do *sid*, a sala (sessão) a que diz respeito, através do *roomID*, a classe do servidor na qual o evento está a ser processado e o tipo de evento que indica a acção decorrida. Neste caso, é indicado que um utilizador (ao qual corresponde o *sid*) se juntou à sessão (identificada por *roomID*).

5.3. Interacção ActionScript e JavaScript

Uma das componentes do trabalho realizado visou também a interacção dos widgets de flash, que oferecem funcionalidades multimédia, com outros desenvolvidos em JavaScript, realizados no âmbito de outra dissertação[21], que disponibilizam

funcionalidades de controlo de sessão, como sincronização de contactos, visualização da lista de funcionalidades disponíveis na sessão (entre as quais podem constar as multimédia oferecidas pelos widgets de flash), etc..

Ao nível da experiência de utilização, na aplicação cliente, este começa por interagir com os widgets de sessão (os desenvolvidos em JavaScript no âmbito de outra dissertação), e a dado momento é dada a opção de escolha da funcionalidade multimédia que se pretende utilizar (*whiteboard, sidehshow* ou áudio e vídeo). Nesse momento é necessário carregar o widget de flash e interagir com ele pedindo-lhe para se ligar ao servidor (OpenMeetings/Red5), disponibilizar a ferramenta pretendida e, mais tarde, desligar e fechar. Para que tal aconteça deverão ser invocadas funções existentes nos widgets de flash, criadas para o efeito. A comunicação no sentido contrário também existe, de modo a poder ser dado *feedback* aos widgets de JavaScript.

Como referido anteriormente, ara que os widgets de flash funcionem é preciso que o dispositivo tenha o *plugin* de flash instalado. Relativamente aos sistemas operativos de dispositivos móveis, alguns já trazem o *plugin* de flash "completo" (versão 10.1) instalado, outros trazem apenas uma versão *Lite* do mesmo. Neste último cenário o flash apenas funciona dentro do *browser*, não sendo possível utilizá-lo fora deste. No entanto, visto que os widgets de controlo de sessão, nos dispositivos móveis, deverão funcionar como aplicações autónomas, fora do *browser*, de modo a aceder a funcionalidades nativas como a lista de contactos, foi necessário considerar dois cenários: a existência da versão completa do *plugin* de flash ou da versão *Lite*. Caso o dispositivo contenha apenas a versão mais leve é necessário que os widgets de flash sejam abertos numa janela de *browser*. Contudo, ao optarmos por esta abordagem perde-se a pretendida interacção entre tecnologias. Esta dificuldade foi minimizada utilizando parâmetros no url que é invocado para o início de uma das funcionalidades que requerem flash. Se tomarmos como exemplo que o utilizador pretende iniciar o *whiteboard*, será aberto o *browser* com o seguinte url:

```
http://192.168.90.7:5080/openmeetings/PUC.swf?
roomid=965&sid=2696b5d5cce1af68e77caf588246a238
&whiteboard=1&whiteboardHeight=100&whiteboardWidth=100
```

Assim, para além da localização do swf, da sala à qual o utilizador se irá juntar e a sua identificação, informa-se também o widget a disponibilizar é o que oferece a funcionalidade de *whiteboard* e quais as suas dimensões.

Já nos dispositivos móveis com a versão completa do *plugin* instalada é possível abrir ficheiros swf fora do *browser* e, nesse caso, os widgets flash podem ser carregados dentro da aplicação e assim conseguir a desejada interacção.

ExternalInterface

ExternalInterface é a API que permite a comunicação entre ActionScript e JavaScript, expondo funções flash para serem invocadas através de JavaScript e permitindo, também, chamar métodos programados nesta última linguagem através de ActionScript. ExternalInterface faz parte da linguagem ActionScript e, como tal, a maior parte do desenvolvimento necessário à comunicação é feito no widget de flash.

Abordemos de seguida os dois cenários em separado: interacção ActionScript → Javascript e JavaScript → ActionScript.

ActionScript → JavaScript

Suponhamos existir uma função JavaScript denominada "ConnectionResult" e que recebe como parâmetro uma String. Para a invocarmos, em ActionScript, é necessário fazer o seguinte:

```
ExternalInterface.call("ConnectionResult", "Success");
```

Com a utilização do método *call* da ExternalInterface poderemos facilmente invocar métodos JavaScript a partir de ActionScript. Como pretende ilustrar o código acima, através deste método é possível informar o widget de JavaScript que o estabelecimento da ligação do widget de flash ao servidor (Red5) foi bem sucedido.

JavaScript → ActionScript

É também importante garantir que é possível invocar métodos ActionScript através de JavaScript. Para que tal aconteça é necessário "registar" os métodos (ActionScript) que queremos expor através de uma *callback*. No caso dos widgets já existentes, uma das funções que foi necessário permitir acesso através de JavaScript é a que faz despoletar a ligação entre o cliente e o OpenMeetings (mais precisamente o Red5). Para tal utilizou-se o seguinte código em ActionScript:

```
ExternalInterface.addCallback("connectWidget", connect);
```

Como referido, através do registo de uma *callback* é possível expor o método que queremos tornar acessível por JavaScript. O código acima descrito pretende sugerir que o widget de JavaScript poderá invocará um método, para ele denominado *connectWidget*, que o widget flash (ActionScript) interpretará como sendo a chamada à sua função *connect*.

6. Testes e Avaliação

Como forma de testar e avaliar a implementação desenvolvida, feitos alguns testes aos widgets desenvolvidos bem como a funcionalidades do servidor que serviu de base ao seu desenvolvimento.

Os testes foram realizados nas seguintes máquinas:

Máquina de Cliente – Desktop:

- CPU – Intel Core 2 Duo @ 2,53GHz
- Memória – 4GB
- S.O. – Mac OS X, 10.6.4

Máquina de Cliente – Terminais Móveis:

- Android 2.1 e 2.2
- BlackBerry OS 5.0.308
- Windows Mobile 6.5
- Symbian^3

Máquina de Servidor:

- CPU – Intel Xeon CPU @ 3.00GHz
- Memória – 512MB
- S.O. – Ubuntu 8.0.4.3 LTS, kernel 2.6.24-24

Os testes realizados e que de seguida serão abordados foram divididos em quatro categorias: testes de funcionalidade, portabilidade, desempenho. Os primeiros permitem atestar o funcionamento dos protótipos realizados. Os segundos têm como função verificar a facilidade com que os widgets podem ser funcionam em diferentes terminais e sistemas operativos e *browsers*. Por último, com os testes de desempenho pretende-se medir o tempo que uma determinada tarefa demora a realizar.

6.1. Testes de Funcionalidade

Nesta secção serão explicados alguns exemplos de utilização dos widgets implementados, ilustrados com imagens.

Os protótipos desenvolvidos foram dois e dizem respeito aos Stream Widgets e Media Widgets. Ambos estão alojados num servidor web e podem ser acedidos através do *browser*. No caso dos primeiros, foi criada uma aplicação flash que incorpora os widgets realizados. Nos segundos, os mesmos são acedidos pela página gerada automaticamente pelo GWT.

Nos pontos seguintes serão apresentados os dois protótipos separadamente.

Stream Widgets

Estes widgets são os que oferecem as funcionalidades de video-conferência, *slideshow* e *whiteboard*. O protótipo, comum a *desktop* e dispositivos móveis (sem alteração de código), foi realizado recorrendo à tecnologia Flex e apresenta ao utilizador, numa barra vertical colocada do lado esquerdo, uma lista com os participantes ligados à mesma sala, através do Red5. Nessa mesma barra, está ainda disponível um espaço onde o utilizador poderá iniciar a sua *webcam* e ver a imagem que está a transmitir. A restante área está destinada às diferentes funcionalidades já referidas.

De seguida são mostradas duas imagens do protótipo. A primeira ilustra uma video-conferência com três participantes (2 clientes em *desktop* e outro num terminal

móvel), onde o utilizador ligado pelo terminal móvel apenas está a visualizar e não a transmitir áudio e vídeo. A segunda mostra apenas a área destinada ao *whiteboard*, num dispositivo móvel, numa sessão com dois participantes e onde o utilizador no *desktop* ia criando desenhos.

Figura 10: Video-conferência com a utilização de desktop e terminal móvel

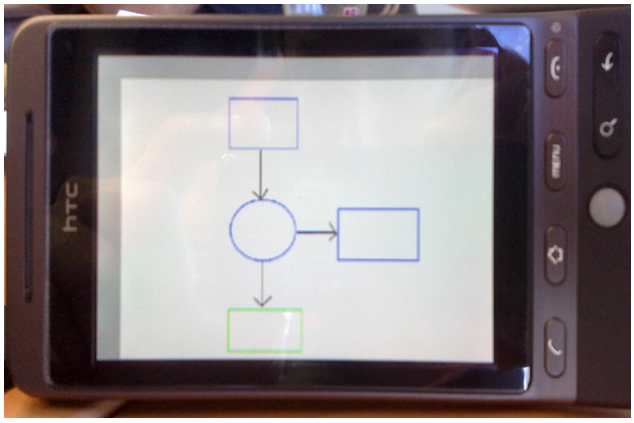

Figura 11: Whiteboard num dispositivo móvel a correr Android 2.1

Media Widgets

Os Media Widgets oferecem as funcionalidades de listagem de ficheiros de uma sessão, bem como de *upload*, *download* e *delete*. Tipicamente, as três últimas funcionalidades enunciadas são chamadas a partir da primeira.

A imagem que se segue mostra a tabela com a listagem dos ficheiros e o widget de *download* incorporado, no fundo:

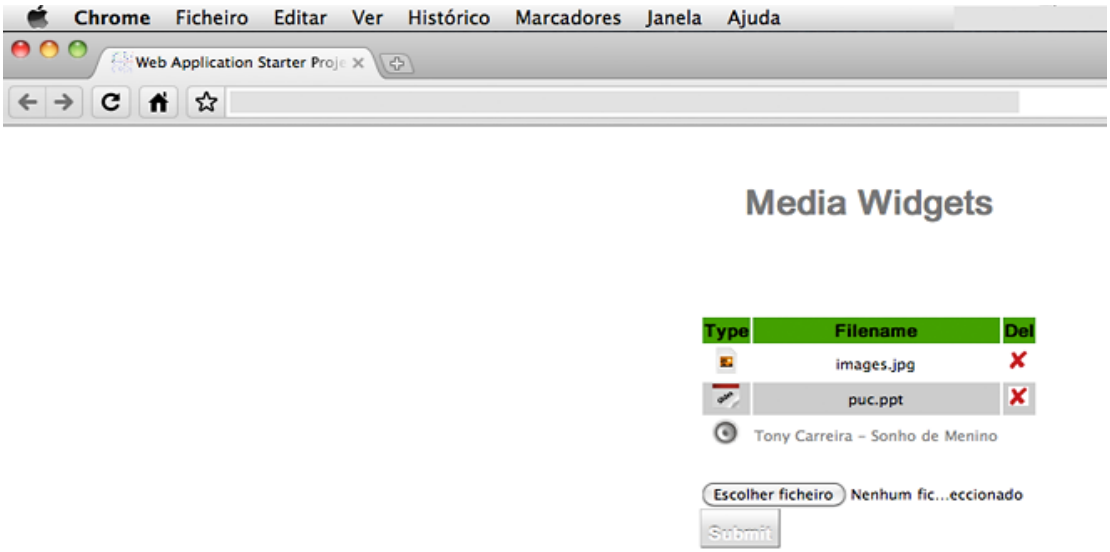

Figura 12: SessionMediaSharing Widget e Upload Widget no browser Chrome

Os widgets de *download* e *delete* podem ser invocados a partir da listagem dos ficheiros. O primeiro, clicando, no nome do ficheiro que se pretende descarregar e o segundo, pressionando a cruz vermelha.

As imagens que se seguem ilustram as janelas de confirmação dos dois widgets, relativamente aos ficheiro images.jpg e puc.ppt.

Figura 13: Download Widget

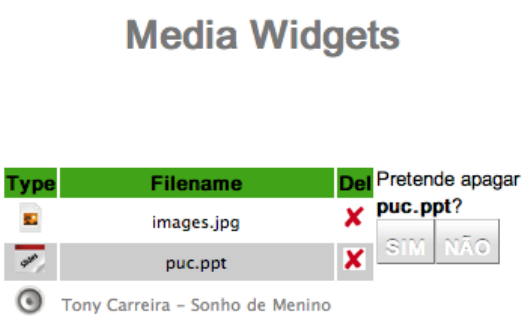

Figura 14: SessionMediaSharing Widget com Delete Widget

Na imagem da esquerda surge a janela de confirmação para o download do ficheiro images.jpg e na imagem da direita aparece a confirmação para a operação de delete relativa ao ficheiro puc.ppt.

Os testes de portabilidade pretendem medir a facilidade com que os widgets podem ser transferidos entre diferentes ambientes (sistemas operativos, *browsers*).

Relativamente aos protótipos desenvolvidos, optou-se por dividir o estudo em duas fases. Na primeira foram testados os widgets em diferentes sistemas operativos de *desktop* e *browsers*. Na segunda fase, fez-se testes em vários sistemas operativos móveis. Em qualquer dos casos foram avaliados diversos *browsers*. Pretendeu-se escolher os S.O. e *browsers* mais representativos do mercado.

No que toca aos testes realizados num *desktop*, foram testados três sistemas operativos diferentes: Mac OS X 10.6.4, Ubuntu 10.04 e Windows 7. Quanto aos *browsers* foram escolhidos o Firefox 3.6.7, Chrome 5.0.375.99 e Opera 10.60 (comum aos três S.O.), bem como o Safari 5.0 para o Mac OS X e o Internet Explorer 8 para a

plataforma Windows. Em qualquer das combinações S.O./Browser todos os widgets (Stream e Media) funcionaram na plenitude das sua capacidades, sem qualquer necessidade de intervenção no código, visto que todos os S.O. suportam o *plugin* de flash e os *browser* apresentam um bom desempenho na execução de código JavaScript.

Já no que diz respeito aos testes realizados nos dispositivos móveis, os resultados foram diferentes. Os sistemas operativos móveis escolhidos foram o Android (versão 2.1 e 2.2), BlackBerry OS 5.0.308, Windows Mobile 6.5 e Symbian^3 (com os *browsers* de origem). Relativamente aos dispositivos, utilizou-se um HTC Hero, Google Nexus One, BlackBerry Bold 9700, Samsung Omnia II e para o sistema Symbian^3 o emulador disponibilizado para o efeito. Em nenhum dos casos houve mudanças do código dos widgets. A imagem seguinte mostra os terminais utilizados:

Figura 15: Dispositivos móveis que serviram para testes de Funcionalidade e Portabilidade

Os resultados obtidos para os diferentes widgets, nos vários sistemas operativos móveis, estão expressos na tabela que se segue:

	Android 2.1	Android 2.2	BlackBerry OS 5.0	Windows Mobile 6.5	Symbian^3
AudioVideo	*	*	-	-	-
Slideshow	√	√	-	-	-
Whiteboard	*	*	-	-	-
MediaSharing	√	√	√	-	√
Download	√	√	-	-	√
Upload	-	√	-	-	√
Delete	√	√	-	-	√

*Legenda: √ - Funciona Totalmente; * - Funciona Parcialmente; - - Não Funciona*

Tabela 2: Validação dos widgets em diferentes terminais móveis

Dos sistemas operativos móveis analisados apenas o Android 2.1 e 2.2 têm *plugin* de flash. Destes, somente a versão 2.2 do sistema da Google contém a versão 10.1, oficial, da Abobe. O Android 2.1 contém uma versão *Lite* do *plugin*.

- Android 2.1 – a funcionalidade de áudio e vídeo apenas funciona parcialmente, uma vez que ainda não foi possível detectar a câmara do telemóvel e transmitir também os dados. Assim, torna-se apenas possível visualizar o que outros estão a transmitir. Relativamente ao *whiteboard* é apenas possível ver (e não participar) porque a escrita de desenhos implica pressionar o ecrã e mexer o dedo. No entanto, este movimento é entendido pelo sistema operativo como o "mover" de toda a aplicação (utilizado, por exemplo, para transições entre aplicações). A funcionalidade de *upload* não funciona de todo, visto que o sistema operativo não

permite chamadas ao gestor de ficheiros e, como tal, não é possível o utilizador navegar pela disco local e seleccionar o ficheiro pretendido.

- Android 2.2 – nesta versão o problema da captura do áudio e vídeo, assim como do *whiteboard*, mantém-se. No entanto, o problema relacionado com o *upload* deixa existir, visto que a última versão do sistema da Google passa a permitir navegar pelo disco local de modo a escolher um ficheiro a ser enviado.
- BlackBerry OS 5.0 – uma vez que este sistema operativo não suporta o *plugin* de flash, os widgets de video-conferência, *slideshow* e *whiteboard* não funcionam. Relativamente aos Media Widgets apenas a listagem de ficheiros funciona. Os restantes não funcionam, possivelmente devido à falta de capacidade do motor de JavaScript incluído no *browser*. Espera-se que com a nova versão do sistema operativo este problema possa ser ultrapassado.
- Windows Mobile 6.5 – tal como no BlackBerry, o Windows Mobile também não suporta o *plugin* de flash e, como tal, os widgets que dependem desta tecnologia estão à partida excluídos de funcionarem neste terminal. No que respeita aos restantes (Media Widgets) também nenhum funcionou. O *browser* mostra uma página em branco e parece não conseguir interpretar o JavaScript. Tal deve-se ao facto do motor de JavaScript utilizado não conseguir interpretar instruções mais complexas.
- Symbian^3 – embora a Symbin Foundation, fundação que promove o desenvolvimento deste sistema operativo, afirme que o Symbian^3 suportará Flash Lite[67] (e como tal se espere um comportamento idêntico ao obtido no Android 2.1), não foi possível testar os widgets dependentes de Flash no emulador. Relativamente aos outros, todos funcionaram plenamente.

6.2. Testes de Desempenho

Os testes de desempenho têm por objectivo medir o tempo necessário para a realização de uma dada tarefa. Foram efectuados os seguintes testes: tempo de propagação da notificação sobre mudança de *slide*, do lado do servidor; tempo que o widget demora a processar a mudança de *slide* desde o momento em que recebe a notificação. Não foi medido o tempo de propagação, por parte do servidor, dos eventos de *whiteboard*, uma vez que o tipo de mensagem enviada (vector de *strings*) e o método utilizado são os mesmos que no caso de *slideshow*. É, portanto, seguro afirmar que os tempos medidos seriam em tudo idênticos.

Foi ainda objectivo medir o desempenho da video-conferência, numa dada sala, aumentado o número de utilizadores ligados e a participarem na mesma. Os testes realizados foram apenas com até 4 participantes e embora a latência medida tenha tipo valores muito baixos (na ordem dos mili-segundos) não é possível inferir alguma conclusão uma vez que não foi possível realizar o teste com muitos participantes (seriam precisos inúmeros computadores, com outras tantas *webcams*).

Mudança de Slide – Propagação

Objectivo

Com a realização deste teste pretendeu-se medir o tempo de propagação de uma notificação de uma mudança de *slide*, pelo OpenMeetings aos seus clientes. Isto é, o tempo que o servidor necessita para comunicar aos participantes, de uma mesma sala, uma mudança de *slide.*

Ambiente de realização

Como cenário criámos uma sala onde variámos o número de utilizadores ligados à mesma, forçámos uma mudança de *slide* e medimos o tempo decorrido desde que o OpenMeetings recebe a notificação até que a envia para todos os participantes. Foram

realizados 5 ensaios para cada número de utilizadores e o tempo apresentado, para cada conjunto de utilizadores, corresponde à média desses mesmos ensaios.

Resultados obtidos

O gráfico que se segue pretende ilustrar os resultados do teste realizado:

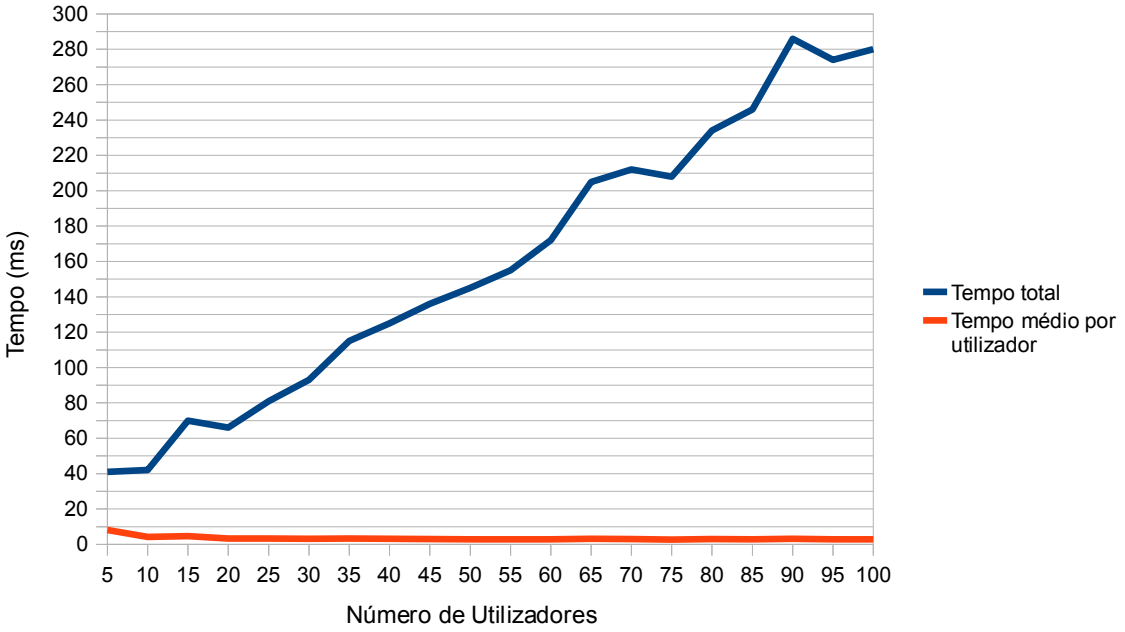

Figura 16: Evolução do tempo de propagação para mudança de slide, no servidor

O envio do evento por parte do servidor é feito percorrendo todas as ligações (da sala) e para cada uma é expedida a mensagem. Uma vez que se trata de um processo iterativo, era já expectável que o tempo total fosse aumentando com o número de utilizadores ligados. Contudo, e apesar do número de clientes ligados ter variado de forma incremental, o tempo total, no pior dos casos, não ultrapassou os 290ms, mesmo quando, numa sala, se encontravam 100 utilizadores ligados.

Mudança de Slide – Concretização no widget

Objectivo

Medido o tempo que o OpenMeetings demora a enviar o evento para o cliente, com este teste pretendeu-se perceber como se comportava o widget no que toca ao tempo necessário para efectivar a mudança de *slide*. Ou seja, o intervalo de tempo desde o instante em que o widget recebe a notificação até que a alteração é, de facto, efectuada.

Ambiente de realização

De modo a testar o que se pretendia, procederam-se a mudanças de *slide*, em 10 diapositivos diferentes e mediu-se o tempo decorrido desde o momento em que o cliente recebe a notificação até que procede à modificação. Foram feitos 5 ensaios para cada mudança de *slide.* Os valores do eixo do x correspondem aos *slides* e o tempo correspondente à média dos 5 ensaios realizados.

Resultados obtidos

Eis os resultados obtidos, mostrados graficamente:

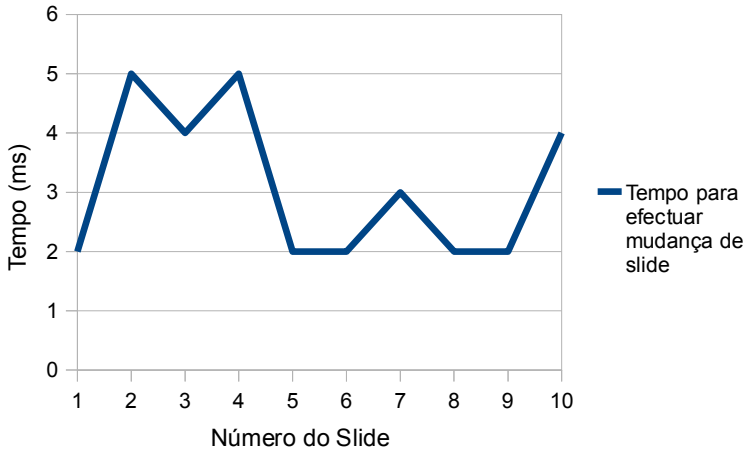

Figura 17: Análise do tempo de mudança de slide, no cliente

Como é possível observar no gráfico, o tempo de processamento para mudança de *slide*, nos 10 testes realizados, variou entre os 2 e os 5 mili-segundos. Parece-nos ser um tempo muito bom para o poder computacional disponível nestes dispositivos.

É expectável que numa utilização real, numa sala com até 100 utilizadores, uma mudança de *slide* surta efeito (em todos participantes) com uma latência de aproximadamente 295ms, o que é bastante aceitável para este tipo de utilizações.

Número de utilizadores ligados na mesma sala

Objectivo

O ActionScript é uma linguagem que não suporta *multithreading*[68] e, como tal, optou-se por simular uma "rajada" de pedidos de ligação e analisar o comportamento do servidor relativamente ao tempo médio gasto para o estabelecimento dessa ligação. Foram, também, registados os pedidos falhados. Entende-se por pedido falhado quando o servidor não consegue ligar um determinado utilizador à primeira tentativa e, por tempo para estabelecimento de ligação, o intervalo de tempo desde o instante em que a o widget faz o primeiro pedido ao servidor até que recebe a mensagem de sucesso ou falha.

Ambiente de realização

Para a realização deste teste, o aumento aumento do número de utilizadores foi gradual. Começou-se por testar 5, depois 10, seguidamente 15 e assim sucessivamente até 100. Para cada número de utilizadores foram feitos 5 ensaios.

Resultados obtidos

Os gráfico que se segue pretende representar os dados recolhidos durante a realização do teste, relacionado o número de ligações com o tempo médio para o estabelecimento das mesmas e o número de ligações falhadas:

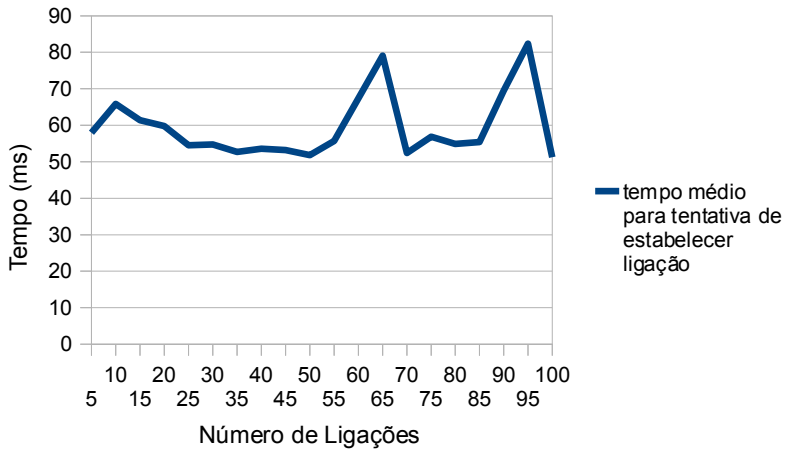

Figura 18: Evolução do tempo médio para estabelecimento de ligação, por número de ligações

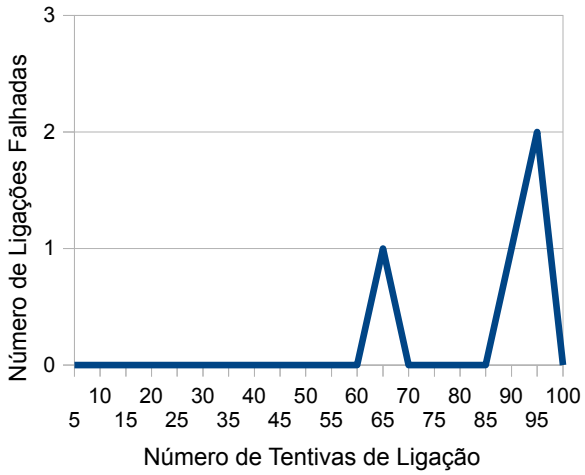

Figura 19: Número de ligações falhadas em função do número de tentativas realizadas

É possível observar no gráfico da Figura 18 que os dois 2 "picos" que se destacam, no que toca ao tempo médio gasto para estabelecer uma ligação, têm correspondência com os "picos" existentes do gráfico da Figura 19. Poderá concluir-se que a demora denotada em dois pontos do primeiro gráfico foi motivada pelas tentativas de ligação fracassadas (uma vez que o servidor, durante um determinado período de tempo fica a tentar

estabelecer a ligação até que desiste). De notar ainda que o número de tentativas de ligação que fracassaram foi escasso, considerando o curto espaço de tempo em que eram feitas. Calculada a média do tempo total para a tentativa de estabelecimento de uma ligação, de todos os ensaios, obtivemos 59,52ms considerando as falhas e 56,78ms excluindo estas. Trata-se de um resultado bastante aceitável para estabelecer uma ligação, considerando, inclusive, que o tempo se mantém estável independentemente do número de ligações tentadas.

Tempo médio de Upload

Objectivo

Um dos testes realizados foi perceber qual o tamanho máximo do ficheiro suportado pelo OpenMeetings aquando do *upload* e de que maneira o tempo (incluindo a escrita em disco do servidor) era afectado com a variação do tamanho. A *servlet* usada para o efeito foi a já referida UploadHandler.

Ambiente de realização

Foram criados vários ficheiros com dados aleatórios de dimensões crescentes em intervalos de 1MByte, utilizando o comando *dd* numa consola UNIX, e medido o tempo necessário para que a *servlet* recebesse o ficheiro e o guardasse em disco. Cada ensaio foi realizado 5 vezes e os tempos apresentados correspondem à média dos valores obtidos.

Resultados obtidos

De seguida pretende-se ilustrar os resultados obtidos através de um gráfico que relaciona o tempo de processamento com o tamanho do ficheiro enviado:

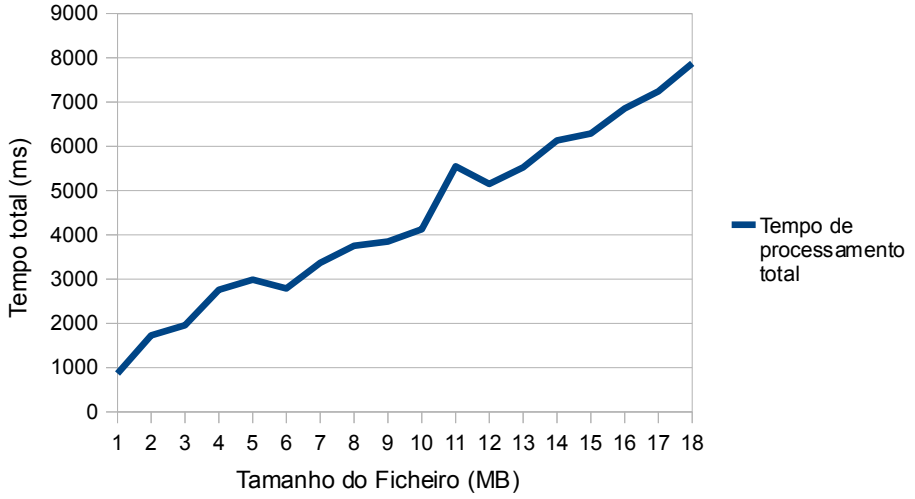

Figura 20: Evolução do tempo de processamento de um ficheiro face ao tamanho do mesmo

Como se pode observar, o tempo cresce de forma linear com o aumento do tamanho do ficheiro, atingindo o valor máximo aos 18MB. Quando foi testado um ficheiro com 19MB o servidor lançou a excepção *Java heap space*. Tal situação carece de ser tratada no futuro. A razão para que isto aconteça prende-se com o facto de a *servlet* guardar, primeiro, os dados recebidos em memória e só depois escrever para disco.

Downloads em paralelo

Objectivo

A realização deste teste teve por objectivo avaliar o comportamento do servidor no que toca ao tempo médio de *download* e ao número médio de descargas, quando submetido a vários pedidos de *download* em paralelo, num determinado período de tempo.

Ambiente de realização

Fez-se variar o número de utilizadores que, durante 2 minutos, descarregaram continuamente ficheiros de 1MB. Simultaneamente, mediu-se a evolução do tempo médio de *download*, por ficheiro e o número total de descargas feitas em cada teste. Mais uma vez, também aqui cada ensaio foi feito 5 vezes e o tempo apresentado corresponde ao tempo médio dos valores obtidos.

Resultados obtidos

Graficamente pretende-se representar a variação do tempo médio e número de *downloads* em função da quantidade de utilizadores a realizar descargas em simultâneo.

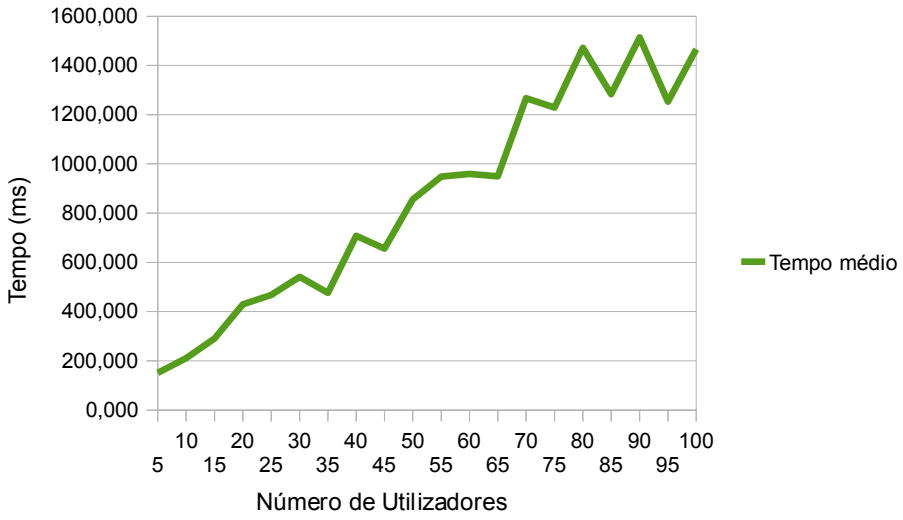

Figura 21: Variação do tempo médio de download em função do número de utilizadores simultâneos

Apesar da existência de algumas oscilações no tempo médio de *download*, é possível inferir uma clara tendência de subida deste com o aumento do número de utilizadores a descarregarem os ficheiros, em paralelo.

Conversão de Slideshow

Objectivo

Uma vez que uma das funcionalidades oferecidas é a de *slideshow* pretendeu-se aferir, sumariamente, qual a capacidade de conversão do servidor, no que toca ao tempo despendido para converter o ficheiro duma apresentação.

Ambiente de realização

Para o efeito criaram-se vários ficheiros *powerpoint*, baseados em texto, onde se fez variar o número de diapositivos por ficheiro e se mediu o tempo de conversão dos mesmos. À semelhança dos testes anteriores, cada ficheiro foi enviado 5 vezes, pelo que os valores apresentados para cada ficheiro correspondem à média dos ensaios.

Resultados obtidos

O resultado obtido expressa-se no seguinte gráfico:

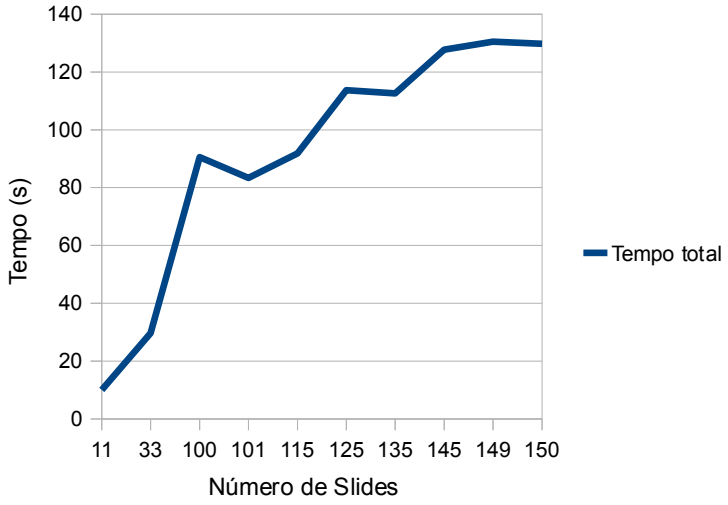

Figura 22: Evolução do tempo total despendido em função do número total de slides a converter

Apesar dos valor médio máximo atingido rondar os 130 segundos, é preciso assinalar que o tempo medido inclui a conversão do ficheiro *powerpoint* para pdf, deste para swf e a

criação de dois conjuntos de imagens (uma por *slide*) de diferentes resoluções. Uma vez que se trata de um processo complexo, pensa-se poder concluir que o tempo apresentado é aceitável. Conclui-se ainda que 150 *slides* correspondem ao valor máximo, uma vez que testado com valores superiores, a conversão falhou. A razão para a falha é desconhecida mas será posteriormente analisada. Constatou-se, ainda, que o tempo médio para o processamento de cada *slide* se manteve constante, independentemente do aumento do número de *slides* do documento.

É também possível deduzir que o tempo necessário para que uma apresentação, de 150 *slides* com 10MB, fique disponível para apresentação (incluindo *upload* e conversão) é de aproximadamente 2 minutos e 20 segundos.

7. Conclusões

O trabalho realizado nesta dissertação teve como objectivo modelar e desenvolver uma framework que tornasse possível e facilitasse a criação de aplicações em contexto Web (cliente e respectivo suporte do lado servidor), que façam uso de funcionalidades colaborativas. Foi também objectivo a possibilidade de suportar os mais variados dispositivos móveis, assim com computadores pessoais ou outros dispositivos como sejam as settop-boxes.

Feita uma análise do trabalho a ser desenvolvido na área da colaboração, nomeadamente ao nível das funcionalidades mais usadas, das tecnologias necessárias e da plataforma em desenvolvimento da PT Inovação, surgiu a motivação para estudar e implementar um mecanismo que ajudasse a desenvolver, de forma fácil e consistente, aplicações que pudessem ser integradas na referida plataforma e assim usufruir de diversas funcionalidades colaborativas, tais como, video-conferência, *slideshow*, *whiteboard* e partilha de ficheiros. A plataforma PUC (Plataforma Unificada de Colaboração) serviu de base ao desenvolvimento do trabalho realizado uma vez que já oferece diferentes recursos passíveis de serem utilizados para o usufruto de diferentes ferramentas de colaboração.

Assim, com este ponto de partida, foram desenvolvidos alguns widgets, utilizando tecnologias como o Flex e GWT, dada a sua difusão e portabilidade. Estes *widgets* permitem oferecer aos programadores suporte para um fácil desenvolvimento de novas aplicações com diferentes funcionalidades de colaboração. Foi ainda necessário estender, do lado do servidor, o sistema de eventos já existente, bem como criar novas *servlets* que dessem resposta às necessidades surgidas.

Relativamente aos widgets inicialmente propostos apenas os de Desktop Sharing e Remote Desktop não foram implementados, por falta de tempo. No entanto, são funcionalidades vistas como trabalho futuro e que serão implementadas. Todos os outros foram desenvolvidos e validados. Os resultados obtidos quanto à portabilidade são os esperados e consideram-se satisfatórios. Constatou-se que no computador pessoal todos os widgets funcionaram a 100% e que nos dispositivos móveis o que mais funcionalidades suportou foi o Android 2.2., onde à excepção do *whiteboard* e da videoconferência, cujo funcionamento foi apenas parcial, tudo o resto foi suportado. No extremo oposto aparece o Windows Mobile 6.5 onde nenhuma widget funcionou.

Apesar de nos testes realizados não ter havido alteração no código dos widgets, quer nos testes feitos no computador pessoal, quer nos terminais móveis, é necessário, no futuro, proceder a algumas alterações de modo a ajustar os widgets às dimensões de um ecrã de um dispositivo móvel. É ainda de salientar que é pouco o código necessário desenvolver para se utilizar os widgets realizados.

Como também foi referido na secção de Testes e Avaliação, será necessário alterar a *servlet* encarregue de receber os ficheiros para que o limite de 18MB seja ultrapassado. A solução passará por ir escrevendo o ficheiro para disco enquanto os dados vão sendo recebidos.

Através da análise de testes de desempenho, como a medição do tempo necessário para o estabelecimento de uma sessão, do tempo de propagação de uma mudança de *slide* ou do tempo despendido para que uma apresentação fique disponível, pode-se esperar uma interacção eficaz e com latências baixas, assim como uma resposta e escalabilidade do servidor que se deve manter dentro do desejável.

8. Bibliografia

[1] J. Grudin, “Computer-supported cooperative work: history and focus,” *Computer*, vol. 27, 1994, pp. 19-26.

[2] P. Wilson and Wilson, P., *Computer Supported Cooperative Work: An Introduction*, Kluwer Academic Pub, 1991.

[3] “ITU Corporate Annual Report 2008.”

[4] A.R. TenCate, “Inspiring collaboration through the use of videoconferencing technology,” *Proceedings of the 35th annual ACM SIGUCCS conference on User services*, Orlando, Florida, USA: ACM, 2007, pp. 335-338.

[5] “Citrix Systems » GoToMyPC by Citrix” Available: http://www.citrix.com/English/ps2/products/product.asp?contentID=13994&ntref=prod_top.

[6] K. Khankan and R. Steele, “Challenges for mobile remote access to desktop and web resources,” *Proceedings of the 3rd international conference on Mobile technology, applications & systems*, Bangkok, Thailand: ACM, 2006, p. 56.

[7] K. Drira, A. Martelli, and T. Villemur, *Cooperative Environments for Distributed Systems Engineering: The Distributed Systems Environment Report*, Springer, 2002.

[8] “Web Collaboration Software” Available: http://www.nefsis.com/Web-Conferencing/collaboration-software.html.

[9] “Adobe - Adobe ConnectNow” Available: http://www.adobe.com/acom/connectnow/.

[10] L. Larson and R. Costantini, *Flash Video for Professionals: Expert Techniques for Integrating Video on the Web*, John Wiley and Sons, 2007.

[11] “Jetty WebServer” Available: http://jetty.codehaus.org/jetty/.

[12] J. Hunter and W. Crawford, *Java Servlet Programming*, O'Reilly Media, 2001.

[13] G. Trapani and A. Pash, *The Complete Guide to Google Wave*, 3ones, Inc., 2010.

[14] “Formare | soluções globais de eLearning e bLearning” Available: http://www.formare.pt/inicio.aspx.

[15] “Medigraf - Soluções Globais de Telemedicina” Available: http://www.medigraf.pt/.
[16] A. Correia, “A Framework for Collaborative Applications,” Dissertação de Mestrado, FCT - UNL, 2010.
[17] P. Taborda, “PLAY: Terminal IPTV para Visualização de Sessões de Colaboração Multimédia,” Dissertação de Mestrado, FCT - UNL, 2010.
[18] B. Ferreira, “Comunicações Colaborativas a la Google Wave,” Dissertação de Mestrado, FCT - UNL, 2010.
[19] “Google Wave” Available: http://wave.google.com/about.html.
[20] “Wave-Protocol - Project Hosting on Google Code” Available: http://code.google.com/p/wave-protocol/.
[21] J. Sousa, “Mobi-Collab Aplicações de Colaboração para dispositivos móveis,” Dissertação de Mestrado, IST, 2010.
[22] “GUI widget - Wikipedia, the free encyclopedia” Available: http://en.wikipedia.org/wiki/GUI_widget.
[23] “Blogger Widgets ~ Blogger Widgets” Available: http://www.bloggerplugins.org/search/label/Blogger%20Widgets?&max-results=5.
[24] “OMTP – Open Mobile Terminal Platform, Defragmentation of mobile phone enablers, BONDI” Available: http://www.omtp.org/.
[25] “BONDI - open source industry collaboration for widget and web technologies” Available: http://bondi.omtp.org/default.aspx.
[26] “The BONDI Software Development Kit” Available: http://bondisdk.limofoundation.org/.
[27] J. Allaire, “Macromedia March 2002 - Requirements for Rich Internet Applications.”
[28] D. McCune and D. Subramaniam, *Adobe Flex 3.0 For Dummies*.
[29] S. Ballerini, “Adobe and Industry Leaders Establish Open Screen Project.”
[30] “Open Screen Project Partners” Available: http://www.openscreenproject.org/partners/current_partners.html.
[31] “Adobe Unveils First Full Flash Player for Mobile Devices and PCs.”
[32] D. Yack, *Silverlight 3 Jumpstart*, We Speak You Learn, LLC, 2009.
[33] “The Official Microsoft Silverlight Site: Get Started” Available: http://silverlight.net/getstarted/.
[34] “Moonlight” Available: http://www.mono-project.com/Moonlight.
[35] “The Official Microsoft Silverlight Site: Silverlight for Mobile” Available: http://silverlight.net/learn/mobile/.
[36] “JavaFX 1.2 SDK Requirements” Available: http://java.sun.com/javafx/1/reference/system-requirements-1-2.html#javafxsdk.
[37] J.L. Weaver, *JavaFX Script: Dynamic Java Scripting for Rich Internet/Client-side Applications*, Apress, 2007.
[38] N. Klein, M. Carlson, and G. MacEwen, *Laszlo in Action*, Manning Publications, 2008.

[39] “OpenLaszlo | the premier open-source platform for rich internet applications” Available: http://www.openlaszlo.org/taxonomy/term/1.
[40] “Adobe Flash Platform | Open Screen Project - About” Available: http://www.openscreenproject.org/about/flash_platform.html.
[41] “Sun / Laszlo Project "Orbit"” Available: http://www.openlaszlo.org/orbit.
[42] “Google Web Toolkit - Google Code” Available: http://code.google.com/webtoolkit/.
[43] “Google Web Toolkit Overview - Google Web Toolkit - Google Code” Available: http://code.google.com/webtoolkit/overview.html.
[44] “Gartner” Available: http://www.gartner.com/technology/about.jsp.
[45] “Forums Sun - JavaFX,” *JavaFX vs JMF* Available: http://forums.sun.com/thread.jspa?messageID=10637588#10637588.
[46] “Vyew's clever desktop sharing to flash player using a Java applet” Available: http://blog.onepixeloff.com/index.cfm/2006/1/12/Vyews-screenrecording-applet.
[47] “FlashLight-VNC” Available: http://www.wizhelp.com/flashlight-vnc/index.html.
[48] “Proof of Concept: Silverlight based VNC client” Available: http://ihatelinux.blogspot.com/2009/05/proof-of-concept-silverlight-based-vnc.html.
[49] “Brendan - Perfil do utilizador” Available: http://www.blogger.com/profile/02621126541372291957.
[50] “RealVNC - Java VNC® Viewer” Available: http://www.realvnc.com/support/javavncviewer.html#1.
[51] “Flex Video Conference” Available: http://www.swfflex.com/blog/1/2007/10/Flex-Video-Conference.cfm.
[52] “Desktop sharing with Red5 and Flash” Available: http://sunil-gupta.blogspot.com/2007/07/desktop-sharing-with-red5-and-flash.html.
[53] “The Official Microsoft Silverlight Site: Accessing Web Camera and Microphone” Available: http://www.silverlight.net/learn/videos/silverlight-4-beta-videos/access-web-camera-microphone/.
[54] “The Official Microsoft Silverlight Site: Whiteboard in Silverlight” Available: http://forums.silverlight.net/forums/t/89669.aspx.
[55] C. Kazoun and J. Lott, *Programming Flex 3: The Comprehensive Guide to Creating Rich Internet Applications with Adobe Flex*, Adobe Developer Library, 2008.
[56] “OpenLaszlo Developers Forums - OpenLaszlo Library” Available: http://forum.openlaszlo.org/showthread.php?t=14007.
[57] “RTMP Specification 1.0.”
[58] A.B. Johnston, *SIP: Understanding the Session Initiation Protocol*, Artech House Publishers, 2009.
[59] H. Schulzrinne, A. Johnston, M. Handley, R. Sparks, J. Rosenberg, J. Peterson, E. Schooler, and G. Camarillo, “SIP: Session Initiation Protocol” Available: http://tools.ietf.org/html/rfc3261.

[60] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne, “RTP: A Transport Protocol for Real-Time Applications” Available: http://tools.ietf.org/html/rfc3550.
[61] C. Perkins, *RTP - Audio And Video For The Internet*, Addison-Wesley, 2003.
[62] J. Lazzaro, “Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport” Available: http://tools.ietf.org/html/rfc4571.
[63] “Explicitly mapping ActionScript and Java objects” Available: http://livedocs.adobe.com/blazeds/1/blazeds_devguide/help.html?content=serialize_data_3.html.
[64] ISO - International Organization for Standardization and ISO - International Organization for Standardization, “ISO - International Organization for Standardization” Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=18902.
[65] “Red5 : Open Source Flash Server” Available: http://www.osflash.org/red5.
[66] “Sun - Interface HttpSession” Available: http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpSession.html.
[67] “Symbian^3 Developer Overview” Available: http://developer.symbian.org/wiki/index.php/Symbian%5E3_Developer_Overview.
[68] A. Harul, “Threads in Actionscript 3” Available: http://blogs.adobe.com/aharui/2008/01/threads_in_actionscript_3.html.

9. Anexo

9.1. Envio de imagem para o Servidor

```
private function doPost(myData:ByteArray, firstTime:Boolean): void
{
	var encoder:Base64Encoder=new Base64Encoder();
	myData.position=0;
	encoder.encodeBytes(myData);
	var url:String=this.buildURL(...);
	var req:URLRequest=new URLRequest(url);
	req.method=URLRequestMethod.POST;
	req.contentType
	req.data=encoder.flush();
	var loader:URLLoader=new URLLoader();
	loader.dataFormat=URLLoaderDataFormat.BINARY;
	loader.addEventListener(Event.COMPLETE, loader_complete);
	loader.load(req);
}

private function loader_complete(event:Event): void
{
	if (this.firstTimePosting)
	{
		this.firstTimePosting=false;
		this.doPost(this.jpgStream, this.firstTimePosting);
	}
}
```

9.2. ProcessBinaryData

```
public class ProcessBinaryData extends HttpServlet {
public ProcessBinaryData(){}
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        HttpSession session = request.getSession() ;
        String sid = "" ;
        boolean savePic = true ;
        if (session.getAttribute("sid") == null)
        {
                sid = request.getParameter("sid") ;
                session.setAttribute("sid", sid) ;
                if (sid == null) { sid = "default" ; }
                String room_id = request.getParameter("room_id") ;
                session.setAttribute("room_id", room_id) ;
                savePic = false ;
        }
        if (savePic)
        {
        if (sid.compareTo("") == 0) {
                sid = (String) session.getAttribute("sid") ;
                long users_id =
                        Sessionmanagement.getInstance().checkSession(sid) ;
                session.setAttribute("users_id", users_id) ;
                long user_level =
                         Usermanagement.getInstance().getUserLevelByID(users_id) ;
                if (user_level > 0)
                {
                        ServletInputStream stream = request.getInputStream() ;
                        BASE64Decoder decoder = new BASE64Decoder() ;
                        byte[] decodedBytes = decoder.decodeBuffer(stream) ;

                        String room_id = (String)session.getAttribute("room_id") ;
                        FileOutputStream fos = new FileOutputStream(Path do write
                        file using System.currentTimeMillis()) ;
                        fos.write(decodedBytes); fos.flush(); fos.close();
                }
        }
}
```

9.3. OMEvent

```
public class OMEvent implements Serializable
{
	public OMEventType eventType;
	public Object roomId;
	public Object userId;
	public Object resourceId;
	public Object uri;

	public OMEvent(OMEventType eventType, Object roomId, Object userId,
		Object resourceId, Object uri)
	{
		this.eventType = eventType;
		this.roomId = roomId;
		this.userId = userId;
		this.resourceId = resourceId;
		this.uri = uri;
	}

	public OMEventType getEventType() { return eventType; }
	public Object getRoomId() { return roomId; }
	public Object getUserId() { return userId; }
	public Object getResourceId() { return resourceId; }
	public Object getURI() { return uri; }
}
```