Universidade de Aveiro
2011
Departamento de Electrónica, Telecomunicações e Informática

Daniel Dos Santos Ferreira

Cloud para comunicações entre instituições médicas

"War does not determine who is right - only who is left."
- Bertrand Russell

Universidade de Aveiro
2011

Departamento de Electrónica, Telecomunicações e Informática

Daniel Dos Santos Ferreira

Cloud para comunicações entre instituições médicas

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Carlos Manuel Azevedo Costa, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Doutor Carlos Manuel Azevedo Costa
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Doutor Rui Pedro Sanches de Castro Lopes
Professor Coordenador do Departamento de Informática e Comunicações do Instituto Politécnico de Bragança

Agradecimentos

Agradeço a minha família o incansável apoio.
Também agradeço ao grupo de bioinformática pela valiosa cooperação.

Palavras-chave

Imagem Médica, DICOM, Sistemas Distribuidos, Cloud Computing, Serviços Web.

Resumo

Ao longo das últimas décadas, os sistemas informáticos que permitem o arquivo e partilha de imagens médicas têm vindo a tornar-se importantes ferramentas de diagnóstico e estudo de patologias, estimando-se um crescimento do volume de informação gerado anualmente de Terabytes para Petabytes. Verifica-se ainda que os equipamentos de aquisição e os locais nos quais se podem encontrar imagens médicas em formato digital têm vindo a tornar-se cada vez mais dispersos. Esta dispersão, associada a diferentes necessidades de fluxo de informação, levantam sérios problemas de organização bem como de acesso integrado aos dados. Nesta dissertação é estudado o uso de tecnologias *cloud computing* com o objectivo de promover a pesquisa e acesso integrado a informação imagiológica dispersas por várias instituições.

Keywords

Medical imaging, DICOM, Distributed Systems, Cloud Computing, Web Services.

Abstract

Over the last decades, the production of digital medical imaging has been increasing. Moreover, the equipments are more accessible and the places where is possible to produce medical images have become more dispersed. This dispersion, associated with different needs of information flow imposes serious problems and challenges, namely issues related with organization and integrated access to data. Here, the information systems that support the storage and share of medical images have become important diagnostic and therapeutic tools. In this thesis, the cloud computing paradigm is studied with goal of promoting the search and integrated access to imagiological information spread over several facilities.

Conteúdo

Lista de Imagens

Lista de Tabelas

Dicionário de acrónimos

ACR - American College of Radiology
AES - Advanced Encryption Standard
API - Aplicational Programming Interface
DICOM - Digital Image and Communication in Medicine
EC2 - Elastic Compute Cloud
GAE - Google App Engine
GQL - Google Query Language
GUI - Graphical User Interface
HTML - Hypertext Markup Language
HTTP - Hypertext Transfer Protocol
HTTPS - Hypertext Transfer Protocol Secure
ISO - International Organization for Standardization
ISP - Internet Service Provider
JPEG - Joint Photographic Experts Group
JSON - JavaScript Object Notation
JVM - Java Virtual Machine
NAT - Network address translation
NEMA - National Eletronical Manufacturers Association
OS - Operating System
PaaS - Platform as a Service
PACS - Picture Archiving and Communication System
REST - Representational State Transfer
S3 - Simple Storage Service
SOAP - Simple Object Access Protocol
SQL - Structured Query Language
SSL - Secure Sockets Layer
TCP - Transmission Control Protocol
TCP/IP - Internet protocol suite
URL - Uniform Resource Locator
VM - Virtual machine
WADL - Web application description language
WADO - Web Access to DICOM Persistent Objects
WAN - Wide Area Network
XML - Extensible Markup Language

1 Introdução

Neste capítulo serão abordados os seguintes aspectos: enquadramento do problema abordado por este trabalho; os factores de motivação para resolução do mesmo; os objectivos da solução apresentada e a estrutura do resto do documento.

1.1 Enquadramento

As imagens médicas desempenham actualmente um papel fundamental nas decisões tomadas em diversos Hospitais e Centros de imagem [1], possuem importantes biomarcadores e fornecem uma descrição da anatomia e dos processos físicos existentes [2]. Estas podem ser geradas por diversas modalidades como por exemplo: tomografia computorizada, ressonância magnética, ultra-som, mamografia [1], entre outras.

Com o aumento da popularidade dos sensores digitais em ciências médicas, a quantidade de informação produzida por estes tem vindo a aumentar de forma exponencial, estimando-se que o volume de informação gerado seja actualmente da ordem dos Petabytes anuais [2], pelo que torna-se cada vez mais necessário implementar estruturas capazes de organizar tal informação de forma eficiente.

Outro aspecto a salientar, prende-se na necessidade do uso de informação de diferentes entidades para o mesmo paciente. Esta necessidade pode ser consequente de diversos factores como, por exemplo: pela mudança do paciente de instituição médica [3], pela falta de capacidade para diagnosticar alguma patologia na instituição de origem sendo necessário efectuar exames numa entidade externa. Assim, as instituições de cuidados de saúde têm vindo a tornar-se fortemente ligadas e cooperantes. No entanto, alguns problemas surgem como demonstrado no seguinte exemplo: uma instituição (A) precisa que um dado exame seja efectuado noutra instituição (B) e interpretado numa terceira instituição (C). Neste exemplo podem existir problemas quer no que diz respeito à comunicação do pedido de exame quer na cobrança dos serviços prestados por cada uma das instituições, bem como na própria comunicação do exame de B para C [4].

Este trabalho focou-se em colmatar necessidades existentes na transmissão de imagens médicas que são o resultado típico de exames efectuados. Para isso foram utilizadas um conjunto de tecnologias WEB e foi desenvolvido um módulo para integrar na plataforma Dicoogle, um sistema *open-source* que permite tratar da transmissão de imagens médicas.

1.2 Motivação

A plataforma Dicoogle usa uma abordagem genérica para indexação e envio de conteúdos. Tem por objectivo minimizar o esforço de configuração e de implantação, permitindo desta forma diminuir os custos, quer de instalação quer de manutenção, face aos tradicionais repositórios de imagem médica. Isto torna-o apelativo do ponto de vista económico. Para transmissão de dados é usada uma tecnologia ponto-a-ponto que torna possível a comunicação dentro de uma rede hospitalar local de forma eficiente [5]. No entanto, ainda não permitia que diversas instituições de cuidados saúde comunicassem entre si de forma transparente.

Dado o carácter confidencial da informação manipulada por este tipo de instituições, estas encontram-se tipicamente protegidas por *firewall*, com todo o tráfego para fora da rede da unidade hospitalar a passar por elas e permitindo apenas um grupo restrito de protocolos para o exterior. De forma a garantir a segurança da informação existente são colocadas medidas ainda mais restritivas a comunicações iniciadas a partir do exterior para dentro da rede da instituição. Para dotar a plataforma Dicoogle de mecanismos de comunicação que permitam a transmissão de conteúdos entre instituições, sem necessidade de configurações adicionais, é necessário recorrer a um serviço externo que seja reconhecido pelas diversas instituições que desejam comunicar. Assim, todas as comunicações entre instituições passariam a depender deste serviço. Uma representação do tipo de rede a tratar pode ser encontrada na Figura 1.

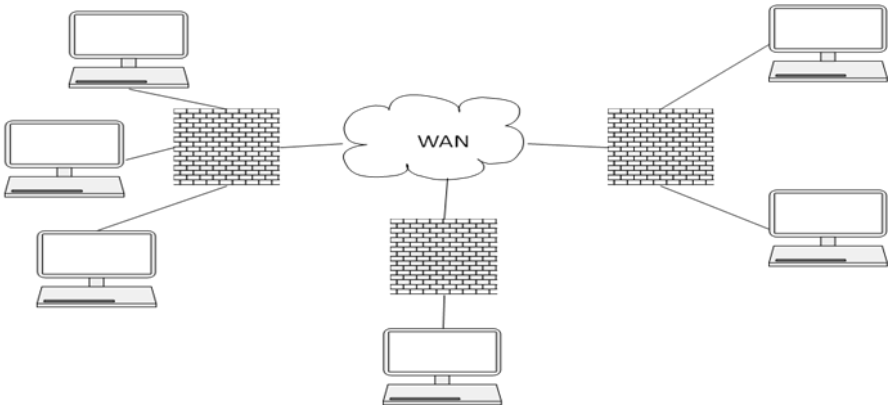

Figura 1. Arquitectura Rede WAN

Uma vantagem de um serviço desta natureza é o facto de constituir um ponto de referência o qual permite, para além de comunicações inter-institucionais, sessões de teletrabalho, pois os especialistas podem ter acesso remoto a exames efectuados pelos seus pacientes sem necessidade de penosas configurações.

Uma vez que não se sabe *a priori* qual a quantidade de tráfego que será necessário suportar por tal serviço, torna-se praticamente impossível dimensioná-lo sem que se corra o risco de estar a subestimar o volume de tráfego gerado por este, ou a sobrestimar. Ambos os casos podem contribuir de forma negativa para o sucesso do serviço estabelecido, quer pela necessidade de pagar por recursos que não são usados, quer por se subestimar os recursos necessários resultando num serviço sem a qualidade desejável para o utilizador final.

Uma possível solução para os problemas de dimensionamento pode ser *Cloud Computing*. Trata-se de um paradigma emergente que tem como principal vantagem a capacidade de dotar os clientes de serviços com os recursos dinâmicos necessários para que estes possam fazer face a eventuais picos de utilização. Além do mais, estes recursos são normalmente monitorizados pelos fornecedores de serviços *cloud* para que eventuais falhas sejam minimizadas [6]. Os fornecedores destes serviços garantem ainda um certo nível de resistência a falhas. Assim, o paradigma *cloud* torna-se assim ideal para o desenvolvimento do serviço pretendido.

1.3 Objectivos

Este trabalho teve como objectivo o estudo de várias plataformas *cloud* no contexto de partilha de imagens médicas, bem como dos serviços por elas disponibilizados. Pretendeu-se efectuar uma escolha apropriada das tecnologias para dotar a plataforma Dicoogle de recursos que permitam às instituições médicas comunicar de forma transparente através de canais típicos da Internet. Dada a rápida evolução das tecnologias envolvidas no sistema apresentado, optou-se por uma abordagem iterativa em que as funcionalidades do sistema foram incrementadas ao longo de sucessivas iterações. Dado o carácter crítico e confidencial da informação a transportar, são também apresentados mecanismos para tornar a arquitectura mais robusta e menos sujeita a erros de transmissão, bem como mecanismos para manter a integridade e confidencialidade dos dados.

1.4 Estrutura

No capítulo 2 é descrito o estudo de algumas tecnologias, dando particular relevo ao trabalho pré-existente, nomeadamente a estrutura do Dicoogle e os serviços *cloud* disponíveis que possibilitam a criação do serviço de reencaminhamento.

No capítulo 3 procede-se ao levantamento de requisitos e, em última análise, as tecnologias escolhidas. É descrita uma proposta de arquitectura e de implementação usada durante o desenvolvimento deste módulo. São também descritas as diversas iterações do desenvolvimento do módulo apresentado.

No capítulo 4 serão apresentados resultados experimentais que permitiram avaliar a arquitectura proposta.

No capítulo 5 serão apresentadas as conclusões do trabalho e descritas algumas indicações de trabalho futuro face aos requisitos existentes.

2 Estado da arte

Neste capítulo são descritas algumas plataformas *cloud computing* e tecnologias usadas durante a elaboração deste trabalho. São também descritos alguns conceitos pertinentes relacionados com distribuição de imagem médica. No final deste capítulo será ainda descrita a estrutura que serviu como base ao desenvolvimento do módulo apresentado neste trabalho.

2.1 Cloud Computing

Tradicionalmente, os laboratórios adquirem servidores locais para tarefas que exigem cálculos intensivos ou manuseamento de grandes volumes de informação. Por exemplo, na área biomédica a colaboração interdisciplinar sobre Internet torna necessário que os laboratórios adquiram a infra-estrutura necessária a esta interacção, verificando-se que a quantidade de máquinas destinadas a partilha de dados entre esses laboratórios tem vindo a crescer [7]. Esta partilha de recursos possui frequentemente períodos de maiores e menores taxas de transferência de dados dependendo, por exemplo, da altura do dia. Tal situação implica grandes variações nos requisitos da infra-estrutura necessária à operação.

Desta forma, para fornecer uma qualidade de serviço aceitável para os objectivos identificados, estas infra-estruturas têm de ser dimensionadas para os piores cenários de utilização, incluindo requisitos de espaço, refrigeração, energia, para além de negociação sobre os protocolos e como configurar firewalls entre instituições. Para estas instituições, *cloud computing* pode representar um novo paradigma no seu negócio, atendendo que os fornecedores destes serviços geralmente alugam pelo menos o hardware/infra-estrutura existente e asseguram a manutenção destes aos seus clientes [7].

Cloud Computing continua em evolução [8] e existem múltiplas definições para este conceito [8, 9, 10, 11, 12, 13]. Uma definição pode ser a seguinte: tipo de sistema paralelo e distribuído, com um conjunto de recursos possivelmente virtualizados, facilmente usáveis e acessíveis, incluindo hardware, plataformas de desenvolvimento e/ou serviços. Estes recursos podem ser dinamicamente reconfigurados e ajustados com base num contrato de prestação de serviço [8, 11]. Como sistema distribuído entenda-se uma colecção de computadores que parecem, do ponto de vista do utilizador, um único sistema coerente [6]. Esta reconfiguração dinâmica de recursos tem por base um modelo de pagamento associado à utilização de recursos [8]. Por exemplo, uma aplicação que funcione sobre *cloud* pode utilizar os recursos numa base *pay-as-you-go*, ou seja, um cliente pode iniciar o seu projecto numa escala pequena e dinamicamente ajustar a utilização dos recursos consoante o seu volume de negócios ou fazer face a potenciais picos de utilização do sistema. Assim, é possível rapidamente, aumentar ou reduzir a quantidade dos recursos reservados e quem efectivamente precisa de ter uma infra-estrutura preparada para este tipo de problema são os fornecedores de serviços *cloud*, transferindo assim o risco do cliente para o fornecedor.

O paradigma *cloud* tem sido amplamente adoptado quer por ser fortemente promovido por líderes da indústria tais como Microsoft, Google, e IBM, quer devido à maturidade das tecnologias por ele usadas e disponibilizadas [9]. No entanto, não foi possível fornecer uma definição formal o que aponta para uma tecnologia relativamente recente, apesar das tecnologias que suportam este paradigma serem maduras, o que parece ser algo paradoxal. Diversos investigadores apontam *Cloud Computing* como uma continuação de *Grid Computing* [12]. Esta relação é bastante evidente em [14], um artigo de 2003 no qual é descrita uma arquitectura *Grid* em que os recursos são pedidos dinamicamente consoante as permissões dos utilizadores. Neste artigo foram ainda apontados alguns requisitos para o ambiente de execução:

- **Protecção:** deve proteger-se o trabalho do utilizador.
- **Controlo do uso de recursos:** os proprietários dos recursos devem ser capazes de controlar a quantidade de recursos consumidos.
- **Autorização:** têm de se assegurar que as autorizações são definidas.
- **Autenticação:** as operações de gestão devem ser registadas de forma segura.

Os dois primeiros requisitos em *Cloud Computing* são geralmente alcançados com aplicações que virtualizam recursos físicos como, por exemplo: Xen, Kvm, Vmware Virtualization [15, 16, 17]. Com estas aplicações os sistemas operativos executados pelo cliente do serviço *cloud* acedem aos recursos virtuais como se de recursos físicos se tratassem. Quer o paradigma *Grid*, quer o *Cloud*, partilham uma abordagem orientada a serviços e alguns dos seus utilizadores são também os mesmos como, por exemplo: Investigadores que realizam cálculos paralelos pouco acoplados entre si. Existem duas diferenças chave entre os dois paradigmas [12]:

- Os sistemas *Grid* são desenhados para que cada pedido do utilizador possa consumir grandes fracções dos recursos existentes. Os sistemas *Cloud* limitam frequentemente o tamanho de um pedido individual a uma pequena fracção da capacidade total disponível e são projectados para suportar um grande número de utilizadores.
- A Computação *Grid* adoptou uma abordagem baseada numa camada intermédia de software que promove a associação entre recursos que colaboram mas estão separados, existindo a noção de domínios administrativos. Os serviços em *Cloud* não estão associados até à data, ou seja, um sistema *Cloud* é tipicamente gerido por uma única entidade (potencialmente grande) com autoridade administrativa, uma configuração uniforme, políticas de programação normalizadas, entre outros.

Ambos assentam em tecnologias maduras e confiáveis consistindo pois uma alternativa para a implementação do serviço de reencaminhamento (*Relay*) [9].

Uma vez que *Cloud Computing* tem sido amplamente apoiada e comercializada por líderes da indústria [9], isto potencia que serviços baseados neste paradigma sejam largamente usados pelo público em geral, face as tradicionais *Grid* que têm sido mais usadas em ciência [11].

Os serviços *Cloud Computing* geralmente assentam numa economia de escala, quando comparados com *datacenters* científicos. Nestes estima-se que a percentagem de utilização seja bastante baixa, aproximadamente 15-20%, enquanto nos sistemas *cloud* actuais este valor é da ordem do 40%. Outra vantagem de economia de escala reside na possibilidade de posicionamento os servidores da infra-estrutura *cloud* em regiões geográficas estratégicas onde o preço da energia é mais baixo. *Cloud Computing* possui assim aspectos económicos bastante atractivos que têm despertado um vasto interesse [7].

Existem três grandes classes de fornecedores *Cloud Computing* que se designam por *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) e *Software-as-a-Service* (SaaS) [18] descritas de seguida.

2.1.1 Insfrastructure-as-a-Service

Trata-se de arrendar uma infra-estrutura informática como um serviço junto de um fornecedor de serviços *cloud*, de uma forma bastante flexível. Dentro deste tipo de serviços pode-se distinguir o que é conhecido como *Physical Resouce Set* (PRS) e *Virtual Resource Set* (VRS), sendo que o PRS possui uma *Aplicational Programming Interface* (API) dependente do recurso físico sem muito software adicional o que se torna uma abordagem bastante leve quando comparada com VRS em que o recurso é virtualizado. Este último permite uma interface unificada mas geralmente sujeita a mais redundância como, por exemplo, para armazenamento de dados ou para uso de interfaces de rede [10]. Os clientes deste tipo de serviço não controlam fisicamente os recursos disponibilizados mas sim toda a camada de software a partir do sistema operativo. Assim, os serviços IaaS não têm uma grande necessidade de investimento inicial e apresentam um risco muito menor para os seus clientes, face às infra-estruturas locais mais tradicionais [19].

2.1.2 Platform-as-a-Service

Este tipo de plataformas fornece um conjunto de funcionalidades sob a forma de serviços bem definidos. Estes ajudam a concentrar os programadores na lógica de negócio, libertando-os de

preocupações com infra-estrutura (física ou virtual) [8]. Assim, os clientes PaaS são capazes de aproveitar a qualidade dos serviços disponíveis.

Estes sistemas restringem geralmente a funcionalidade da aplicação. Por exemplo, apenas disponibilizam comunicação HTTP/S, ou um número de instruções executadas por pedido limitado para garantir respostas curtas, ou um acesso limitado ao sistema de ficheiros, com o objectivo de proteger a estabilidade do sistema como um todo [18]. Exemplos de plataformas com tais restrições são a Google App Engine (GAE) e a AppScale.

2.1.3 Software-as-a-Service

Qualquer Software que execute sobre *cloud* e forneça funcionalidades destinadas ao utilizador final é designado SaaS [10].

Os seus utilizadores podem assim usar aplicações sem ter de as instalar nem executar nas suas máquinas locais, visto que os fornecedores SaaS disponibilizam software como um serviço sobre rede [20]. SaaS pode assim ser visto como uma nova forma de distribuição de software [19]. Exemplos deste tipo de serviço são o Google Docs, o Office Live, o Dropbox e o Box.net.

2.2 Plataformas Cloud

Actualmente existem diversos fornecedores de serviços Cloud comerciais e algumas iniciativas *open-source* que têm vindo a tornar este paradigma interessante do ponto de vista económico, bem como uma subárea dos sistemas distribuídos ideal para investigação [11, 18]. Diferentes fornecedores *cloud* oferecem diferentes tipos de serviços e recursos [7].

De seguida serão descritos alguns fornecedores de serviços comerciais, bem como sistemas provenientes de iniciativas *open-source* que têm sido alvo de avaliação e investigação científica [6, 8, 11, 18, 19].

2.2.1 Amazon Web Services

Amazon é considerada um dos maiores fornecedores de serviços *cloud*. A sua plataforma Amazon Web Service (AWS) [21] oferece uma ampla gama de serviços distribuídos, escaláveis, dos quais se podem destacar os seguintes [10]: Elastic Compute Cloud (EC2), Simple Storage Service (S3), SimpleDB, Simple Queueing Service(SQS), Elastic Load Balancing, CloudWatch e Auto Scaling.

EC2 é um serviço que fornece acesso a diferentes tipos de imagens de máquinas virtuais Xen. O S3 fornece um serviço de armazenamento que permite dados persistentes e dada a importância que estes dados normalmente têm para os clientes, possui também mecanismos de redundância e recuperação de falhas. O SimpleDB proporciona funcionalidades como linguagem para pesquisas. O Simple Queueing Service permite enviar mensagens entre diferentes máquinas virtuais. O Auto Scaling é um serviço para lançar ou terminar instâncias EC2 de forma automática, baseado nas configurações do utilizador.

De salientar que a plataforma AWS se trata de um serviço do tipo IaaS, que tem sido alvo de estudos da comunidade académica pelos serviços por ele fornecidos [22, 23], com vista a uma análise comparativa em relação aos tradicionais *datacenters* que geralmente existem em diversas unidades de investigação [11]. Este tipo de análise é particularmente interessante para investigação pela forma de pagamento presente neste serviço, uma vez que o custo de usar uma máquina virtual durante 1000 Horas é o mesmo que usar 1000 máquinas virtuais durante uma hora, aspecto atractivo para problemas em que seja possível uma implementação de resolução paralela [6].

2.2.2 Microsoft Windows Azure

Como o próprio nome indica, a Microsoft Windows Azure [24] trata-se da solução *Cloud Computing* fornecida pela empresa Microsoft. Esta plataforma quando comparada com as restantes encontra-se num ponto intermédio no que diz respeito a flexibilidade e conveniência de programação. Sendo comummente descrita como uma plataforma (I/P)aaS. Os utilizadores têm

possibilidade de escolher a linguagem de programação, não sendo contudo possível controlar o sistema operativo.

As aplicações são desenvolvidas recorrendo ao uso de bibliotecas .NET. Estas fornecem certo grau de configuração automática e resistência a falhas, bem como escalabilidade. No entanto, é necessário declarar explicitamente algumas propriedades para fazê-lo.

Esta plataforma ao nível mais abstracto é composta por três componentes:

- **Windows Azure:** Fornece um ambiente para executar aplicações e guarda dados nos datacenters Microsoft.
- **SQL Azure:** Trata-se da versão Microsoft SQL server para o paradigma *cloud*, que permite usar bases de dados relacionais nesta plataforma.
- **AppFabric:** O seu objectivo é tornar a colaboração entre aplicações possível [6, 25].

2.2.3 Google App Engine

Google App Engine [26] trata-se de uma plataforma PaaS, que fornece serviços úteis para pessoas ou empresas que pretendam criar aplicações Web a partir do zero sem necessitarem de se preocuparem com a infra-estrutura.

Inclui escalonamento e balanceamento de carga automáticos dos pedidos existentes sem qualquer necessidade de configurações adicionais [19]. Limita o tempo máximo de cada pedido efectuado pelos clientes da aplicação a 30 segundos e não permite *threads* de execução lançadas pelo cliente de forma explícita embora internamente os seus serviços usem *threads*. Oferece uma ampla gama de serviços entre os quais se destacam: Datastore, Blobstore, Mail, Memcache, Task Queues. O Datastore fornece uma forma escalável de armazenamento persistente de entidades com um tamanho de máximo 1MB e possui funcionalidades para pesquisas baseadas em linguagem GQL (Uma linguagem similar a SQL, para obter entidades). O Blobstore permite armazenar objectos denominados Blobs, os quais podem ter um tamanho máximo de 2GB. O Mail oferece uma forma de enviar e receber emails. O Memcache representa uma memória volátil distribuída, permitindo velocidades de acesso muito maiores às que é possível obter com Datastore. As entidades podem ter no máximo 1MB o que, infelizmente, é bastante limitador em termos de capacidade e não existem quaisquer garantias em relação à permanência dos dados neste serviço. Através do serviço Task Queue as aplicações em App Engine podem executar processamento em *backgroud* inserindo tarefas numa fila. O tempo de execução de cada uma destas tarefas não deve exceder os 10 minutos, momento em que é disparada uma excepção que termina a execução da tarefa a decorrer.

Google App Engine encontra-se internamente organizado em vários módulos. O módulo App Master possui uma interface para armazenar as aplicações e acesso a servidores de ficheiros, bem como acesso aos módulos App Engine frontend e App Servers. Quando é efectuado um pedido a uma aplicação GAE, este é encaminhado para o módulo App Engine frontend consoante a URL (Uniform Resource Locator) enviada. Este reencaminha o pedido para um dos Application Servers existentes[1]. Quando o pedido chega ao Application Server este trata-o dentro de um ambiente de execução restrito. Este ambiente de execução permite o acesso ao serviços distribuídos Google. Por exemplo, o serviço Datastore usa o sistema de armazenamento de dados BigTable que tem vindo a ser desenvolvido há anos para suportar os serviços Google existentes (Web indexing, Google Earth, Google Finance) [27]. Na Figura 2 encontra-se representada a arquitectura do serviço GAE.

2.2.4 Salesforce

O Salesforce **[28]** trata-se de uma Plataforma SaaS com uma API restrita. Fornece um conjunto de ferramentas para modelar os dados do negócio de uma instituição cliente no seu serviço de *cloud*. Estes passam a estar disponíveis, através do acesso a uma interface configurável, a partir do site do serviço *cloud*, bem com através de *Web Services Simple Object Access Protocol*

[1] Se necessário a própria aplicação também será enviada para o Application Server destino.

(SOAP). Está plataforma permite a uma aplicação servir múltiplos clientes sem necessidade de qualquer configuração adicional pois todo o balanceamento de pedidos é tratado de forma transparente pelo fornecedor de serviços *cloud* [19]. Neste serviço, os dados são guardados numa base de dados partilhada onde cada cliente tem apenas acesso à sua própria informação. Para acesso à base de dados é usada a linguagem Apex.

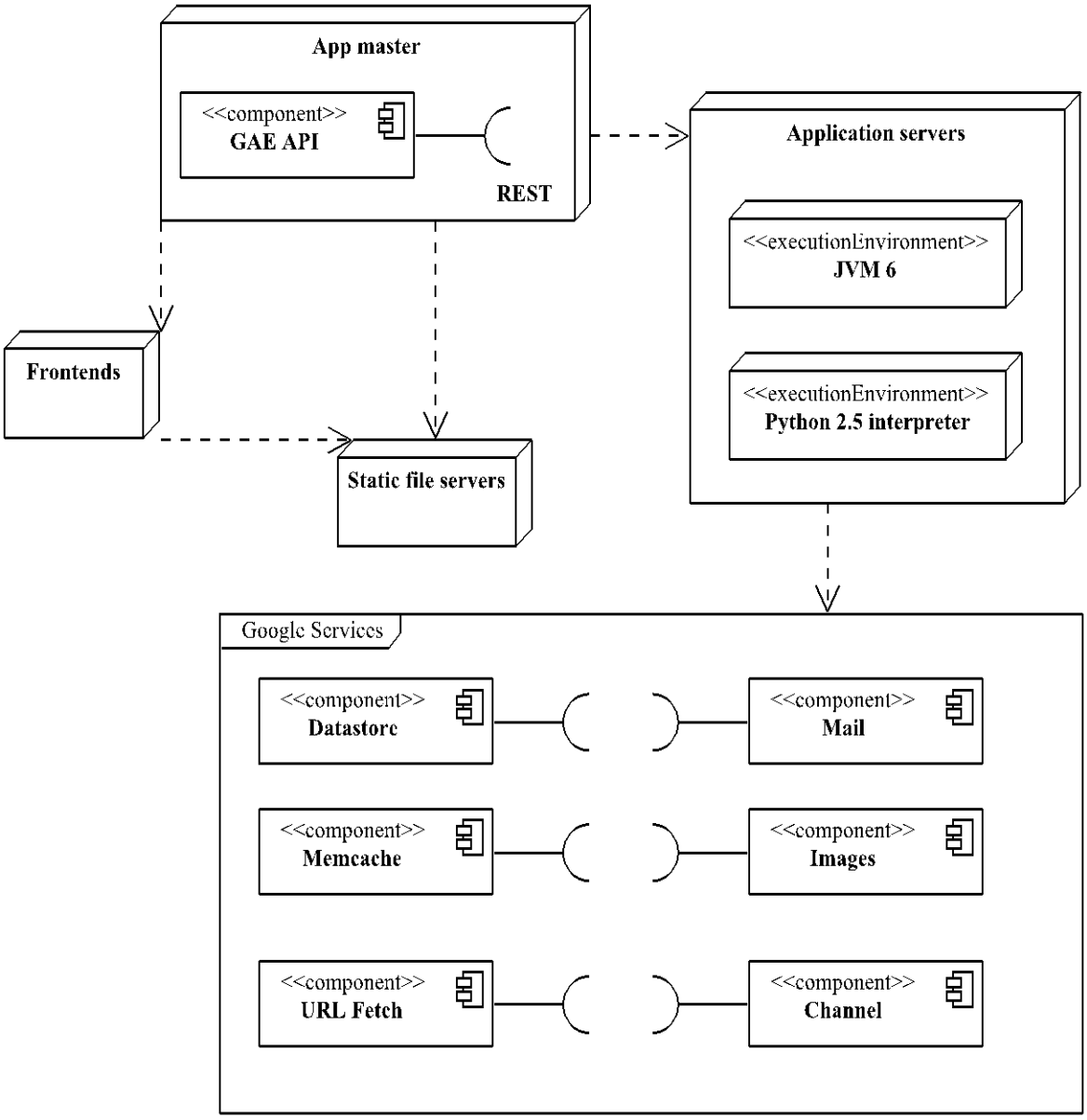

Figura 2. Arquitectura Google App Engine

2.2.5 Nimbus

A primeira *cloud* da Universidade de Chicago foi disponibilizada em 3 de Março de 2008. Contava com 16 nós, cada um com dois processadores AMD64 a 2.2GHz, 4 GB RAM, e 80 GB de disco local. Esta plataforma foi denominada com "Nimbus" [22] e actualmente encontra-se implementada em diversas instituições, permitindo que diversos tipos de investigadores desenvolvam software para *cloud* de forma completamente gratuita [29]. Trata-se de uma plataforma do tipo IaaS e permite a um cliente usar recursos remotos através de máquinas virtuais (VMs). Fornece uma implementação dos Web Services Amazon (EC2) e S3 através de um serviço

denominado Cumulus. Tal permite que os seus utilizadores desenvolvam aplicações EC2 em Nimbus e, caso o sistema Nimbus não corresponda as suas expectativas, podem facilmente migrar para o sistema comercial da Amazon [8]. Na Figura 3, encontra-se representado o diagrama de implantação Nimbus. É possível perceber que não só a interface do serviço *Web Services* S3 se encontra dependente do serviço Culumus, mas também o serviço EC2 encontra-se dependente deste, pois é o local onde os dados das máquinas virtuais são armazenados. Outra importante característica desta plataforma é o facto de se encontrar definida de uma forma modular o que permite uma separação clara das funcionalidades de cada componente, bem como um certo grau de resistência a falhas.

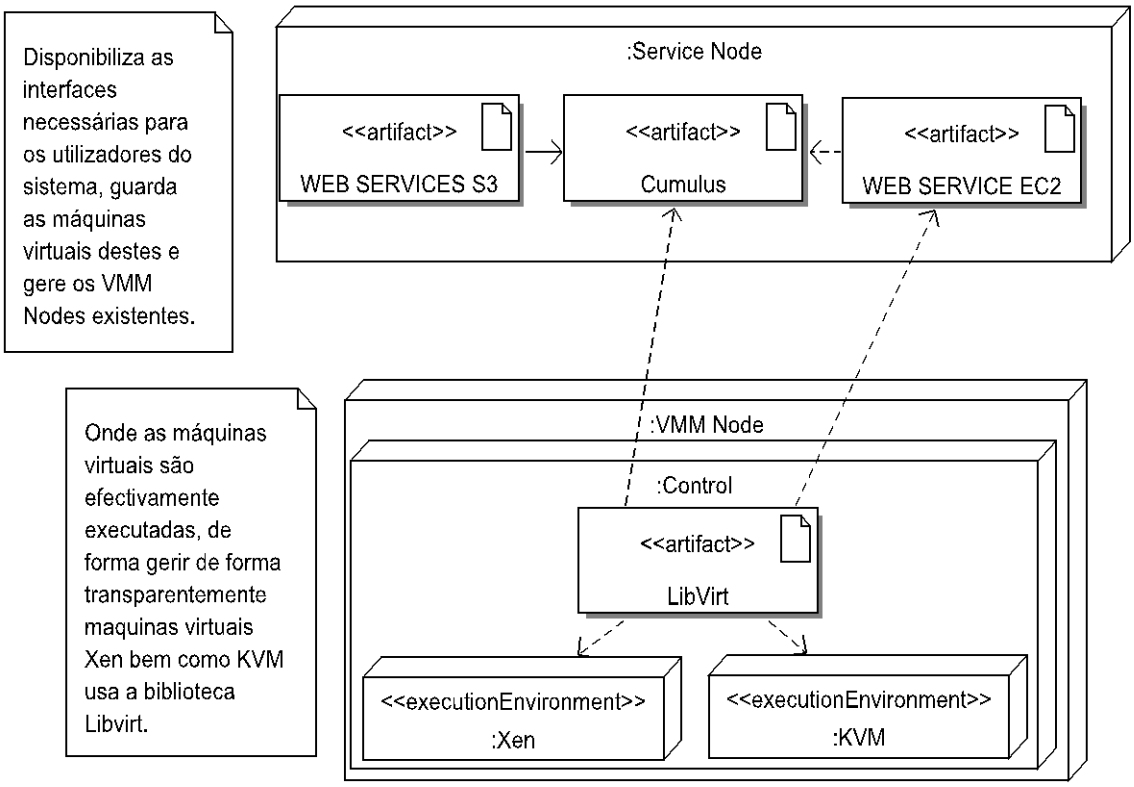

Figura 3. Diagrama Implantação Nimbus

2.2.6 Eucalyptus

Eucalyptus [23] é um *software open-source* desenvolvido na universidade da Califórnia que implementa uma plataforma do tipo IaaS [8]. Esta plataforma disponibiliza os seus serviços através de interfaces definidas pelos serviços Amazon EC2 e S3.

A estrutura fornecida tem sido aproveitada pela comunidade científica e, de forma a promover a sua divulgação, foi criado um serviço *online* no qual é possível efectuar um registo gratuito que permite usar um conjunto de máquinas virtuais de forma algo limitada.

Esta plataforma permite ainda usar diversas ferramentas que foram criadas para Amazon AWS (exemplos: Rightscale, Elastra, HybridFox) [18].

Na Figura 4, é apresentada a arquitectura básica do Eucalyptus que permite executar imagens virtuais da mesma forma que a Amazon EC2. De salientar que nesta representação não está explicita a estrutura Walrus que simula o serviço Amazon S3. A escalabilidade do sistema é assegurada por uma arquitectura modular e hierárquica, que torna a plataforma resistência a falhas e escalável [12].

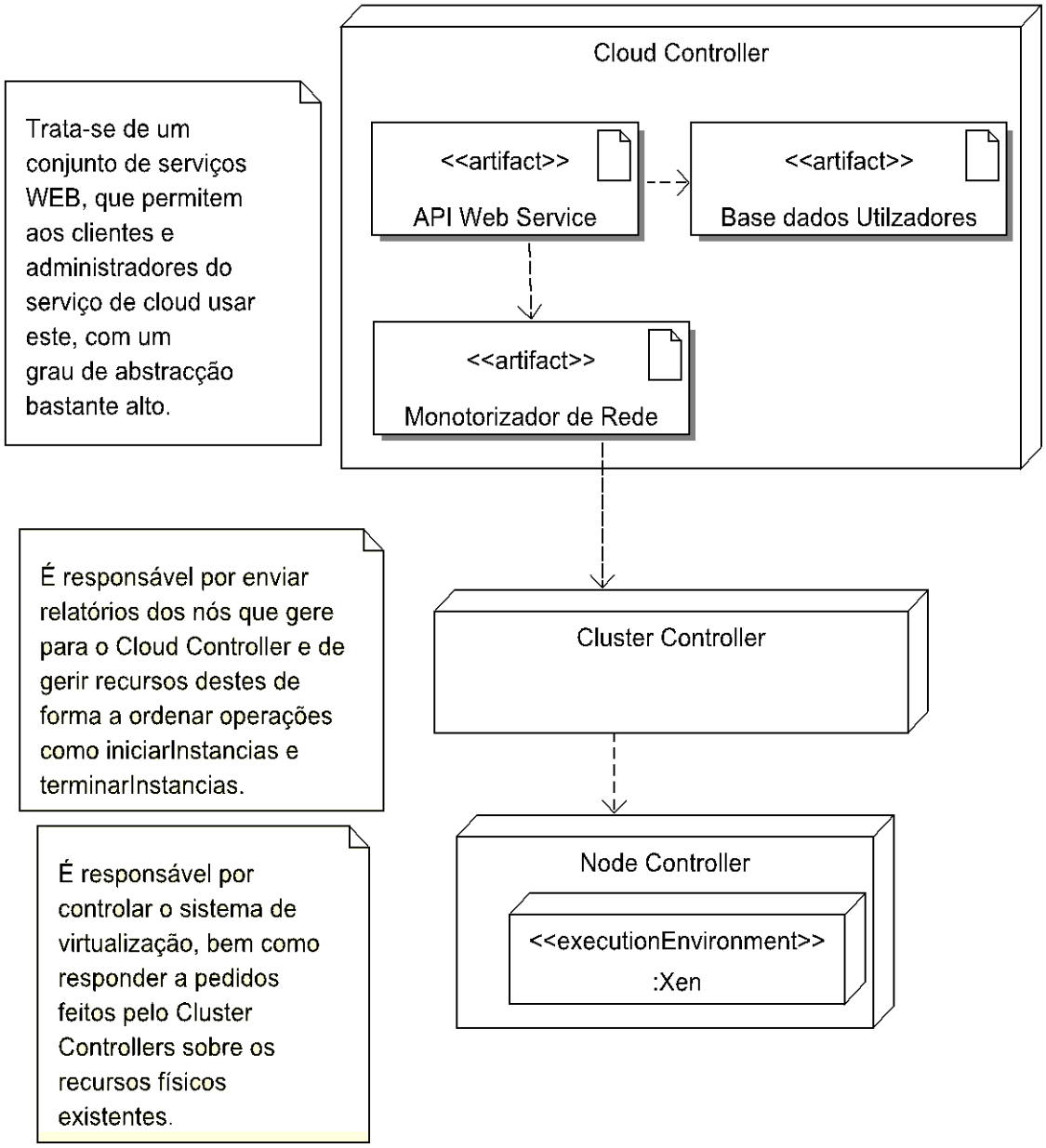

Figura 4. Diagrama Implantação Eucalyptus

2.2.7 AppScale

A plataforma AppScale [30] também está a ser desenvolvida na Universidade da Califórnia. Implementa alguns dos serviços do GAE (por exemplo: Datastore, Memcache) e tem por objectivo fornecer aos investigadores uma estrutura que permita estudar o funcionamento dos sistemas PaaS. Para isso, usa um conjunto de tecnologias livres e uma versão modificada da ferramenta de desenvolvimento disponibilizada pelo Google [30].

De forma a suportar uma execução distribuída de aplicações pelos diversos nós da rede, esta plataforma encontra-se separada em dois grandes componentes:

- **AppLoadBalancer:** No qual é feito o balanceamento e redireccionamento dos pedidos dos clientes.
- **AppServer:** permite um conjunto de funcionalidades definidas para o GAE.

Estes encontram-se visualmente representados na Figura 5.

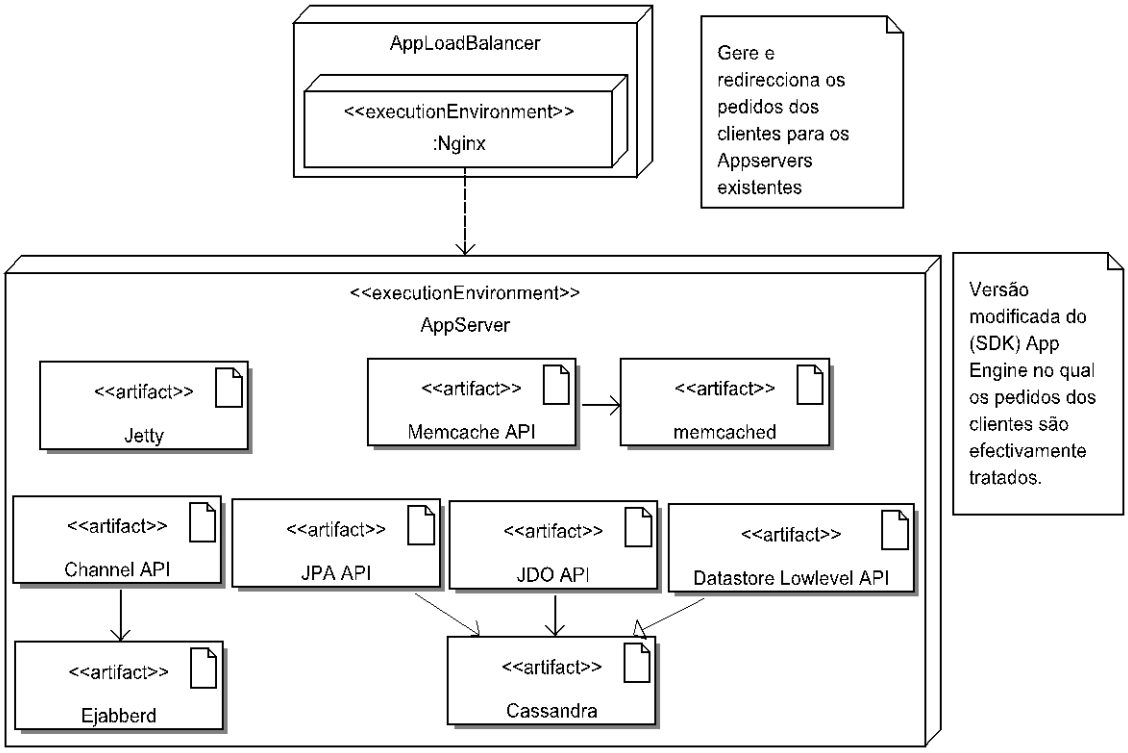

Figura 5. Diagrama de implantação AppScale

2.2.8 Considerações sobre cloud computing

Em suma, pode organizar-se os diferentes serviços analisados em grupos. Existem duas variáveis a considerar entre estes grupos que são a Flexibilidade e Conveniência para o utilizador ou programador. É necessário que exista um compromisso entre ambas pois o aumento de uma tipicamente implica a redução da outra e vice-versa. Os serviços considerados podem ser dispostos como representado no diagrama da Figura 6.

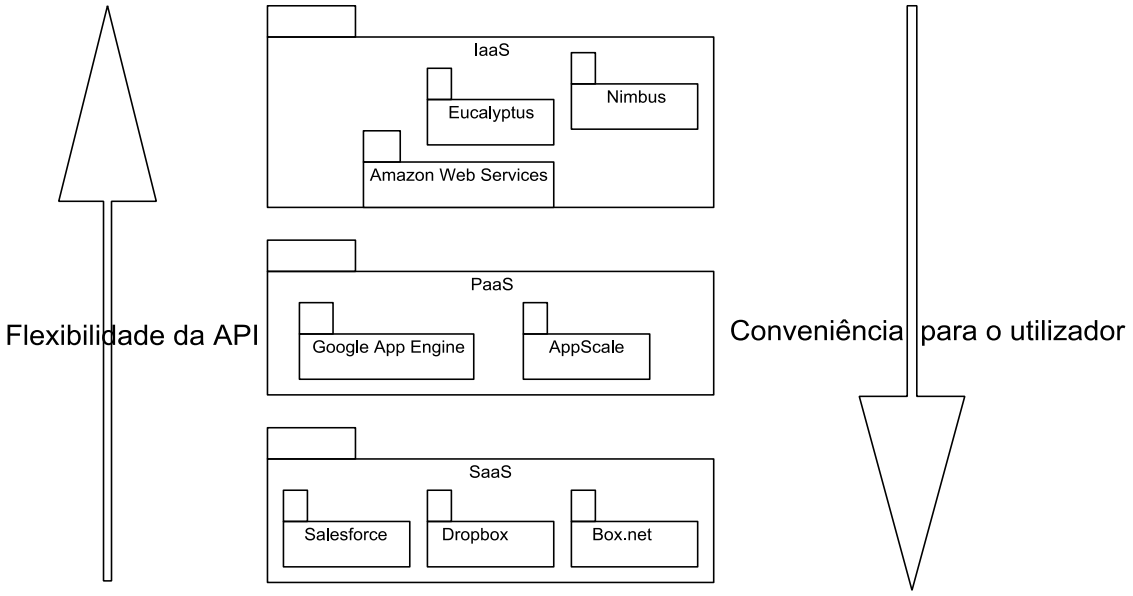

Figura 6. Flexibilidade VS Conveniência dos serviços *Cloud*

Apresentado o paradigma *Cloud Computing* e enunciados alguns dos serviços através do qual é possível usá-lo, vamos agora enumerar algumas das propriedades que deverão ser consideradas num processo de selecção por parte de diversas instituições:

- **Disponibilidade do serviço:** existe um certo risco ao mover uma parte do negócio para uma infra-estrutura *cloud* visto que uma falha no fornecedor de serviços, ou na ligação ao mesmo, pode comprometer o funcionamento do sistema.
- **Dependência da *API cloud*:** este problema assenta na forma como é feito o acesso aos diferentes serviços disponibilizados pelos fornecedores *cloud*, uma vez que o serviço passa a depender das APIs disponibilizadas e actualmente ainda não existe uma normalização das interfaces.
- **Confidencialidade de dados:** outro factor que poderá dissuadir potenciais clientes é o facto de colocarem os seus dados numa entidade externa, o que obriga à criação de mecanismos extra para garantir a confidencialidade dos mesmos, representando pois um certo esforço acrescido quando comparado com uma infra-estrutura própria.
- **Limites na transferência de dados:** actualmente muitas das instituições têm de gerir grandes volumes de informação, pelo que enviar e receber esta informação da *cloud* pode não ser vantajoso do ponto de vista económico.
- **Desempenho Imprevisível:** dentro de um mesmo fornecedor de serviços os servidores de suporte à infra-estrutura *cloud* são geralmente bastante heterogéneos, isto implica que nem sempre a mesma quantidade de recursos utilizados se traduz na mesma performance dos recursos disponíveis.
- **Reputação partilhada:** até que ponto o mau comportamento de um utilizador pode afectar os restantes e o serviço de *cloud* como um todo. Por exemplo, se determinado utilizador recorrer a um serviço de *cloud* para enviar emails não desejados, este acto pode colocar o serviço de *cloud* em *blacklists*, prejudicando potencialmente os restantes clientes [6].
- **Controlo sobre os dados:** os clientes dos serviços *cloud* têm um controlo limitado sobre a infra-estrutura usada para armazenar dados e, por outro lado, os fornecedores de serviços têm privilégios sobre os dados considerados muitas vezes excessivos [20].
- **Segurança:** apesar dos tradicionais servidores localizados numa infra-estrutura local parecerem uma solução muito mais atractiva em termos de segurança, uma análise mais atenta poderá não confirmar tal percepção. Nas infra-estruturas locais é comum encontrar dispositivos amovíveis sem qualquer tipo de protecção, bem como computadores sem as devidas medidas de segurança, o que pode ser problemático manter a devida protecção nos dados. Por outro lado, o facto de uma determinada instituição confiar os seus dados a um fornecedor de serviços *cloud*, tem a vantagem de dificultar os ataques físicos contra determinadas instituições, uma vez que os seus serviços podem estar espalhados fisicamente por diversas máquinas dispersas pela infra-estrutura *cloud* [7].

2.3 Imagem Médica

Com os avanços tecnológicos, os sistemas médicos mais tradicionais baseados em película foram-se tornando obsoletos. Assim, os equipamentos recentes vêm dotados com avançadas tecnologias e sistemas de informação, tendo como objectivo melhorar a qualidade dos cuidados de saúde fornecidos pelas instituições.

Tipicamente, como resultado dos exames radiológicos efectuados são geradas imagens em formatos digitais [1]. Estas possuem importantes biomarcadores e fornecem uma descrição da anatomia dos processos físicos existentes, por isso desempenham um papel fundamental nas decisões tomadas em diversas instituições [2].

2.3.1 DICOM

A partir dos anos 70, os dispositivos para aquisição de imagem médica começaram a proliferar assim como o uso de recursos informáticos para guardar, processar e analisar imagens digitais.

Por isso o American College of Radiology (ACR) e a National Electrical Manufactures Association (NEMA) começaram a cooperar para a definição de uma norma da indústria de imagiologia em 1983. Dois anos mais tarde, em 1985 foi publicada a primeira versão da norma ACR-NEMA 300-1985, onde foram encontrados erros e sugeridas melhorias. Em 1988 foi pública a versão 2.0 da norma que fornecia as ferramentas necessárias para a interacção de dispositivos de imagem médica. Contudo existiam questões de confiança nas comunicações necessitavam de ser normalizadas. Em 1993 foi publicada a terceira versão desta norma que denominada como Digital Imaging and Communications in Medicine (DICOM), versão que tem sido melhorada com regularidade até aos dias de hoje.

Trata-se de uma norma não proprietária que tem sido desenvolvida para responder às necessidades dos fabricantes e utilizadores de equipamentos de imagem médica que pretendia interoperabilidade dos dispositivos. Inclui uma especificação de conteúdo, estrutura de codificação e protocolos de comunicação para troca de imagens biomédicas e informação relacionada. Num ambiente em que existam equipamentos de múltiplos fabricantes, permite evitar interfaces proprietárias para diferentes máquinas, reduzindo assim a dificuldade e o custo de interligação de equipamentos.

Para alcançar tal grau de interoperabilidade, a sintaxe de transferência assenta num conjunto de regras de codificação que são negociadas entre duas aplicações, através de uma sintaxe básica que deve ser suportada por todos os dispositivos que se declarem conforme a norma DICOM. Assim, estes podem iniciar um processo de descoberta de funcionalidades suportadas por ambos para se poderem inter-comunicar inequivocamente entre si. De realçar ainda que, de forma a facilitar a manutenção e evolução da norma como um todo, esta encontra-se dividida em diversas partes [31].

Conformidade

Usar DICOM só por si não garante interoperabilidade, embora torne mais simples alcançá-la. A questão que deverá ser feita por parte dos compradores de equipamentos é como alcançar tal interoperabilidade e o que realmente significa "conforme com a norma DICOM". Dada a complexidade da norma, qualquer implementação desta num dado equipamento irá incluir apenas um subconjunto de funcionalidades existentes. As declarações de conformidade por parte do fabricante deverá identificar claramente que partes da norma suportadas. A parte 2 da norma assegura que estas declarações terão a mesma estrutura global, o que facilita a tarefa por parte de potenciais compradores tendo em vista uma análise de propostas de diversos fabricantes de equipamentos. Também seria razoável para um comprador exigir que os fornecedores expliquem como as suas implementações poderão trabalhar em conjunto com os equipamentos previamente existentes. Isto pode, por exemplo, ser parte de um caderno de encargos num processo de aquisição de equipamento [32].

Assim, DICOM garante até certo ponto a interoperabilidade entre dispositivos, mas é importante saber que nem todo o dispositivo DICOM é compatível com os restantes, a compatibilidade depende dos serviços pedidos e fornecidos.

Formato de ficheiros para troca de dados

DICOM define um conjunto de propriedades para suportar ficheiros sobre diferentes suportes de armazenamento. A parte 10 da norma especifica um modelo para armazenamento de imagens médicas e informações relacionadas com estas. Define como deverá ser feito o encapsulamento de qualquer informação relacionada com DICOM, como será efectuado o

encapsulamento de dados cifrados. Propõem também um serviço de ficheiros DICOM independente do meio físico de armazenamento [33].

Imagens DICOM

A norma específica um conjunto de propriedades tendo em vista a correcta visualização e codificação de imagens. Suporta um dado número de técnicas de compressão como, por exemplo, JPEG com ou sem perdas. Especifica métodos para calibração de dispositivos de apresentação e representação de imagens como, por exemplo, monitores e impressoras [33].

Uma imagem DICOM pode incluir não só a própria imagem, mas também informação relacionada com esta como, por exemplo, detalhes importantes do procedimento de aquisição realizado e texto com interpretações efectuadas.

A possibilidade de enviar, para além das imagens, informação relacionada com estas é uma das características mais importantes que distingue a norma DICOM de muitas outras que se encontram, por exemplo, limitadas apenas aos dados da imagem [31].

Acesso Web para Objectos Persistentes DICOM (WADO)

A última parte da norma introduzida foi o serviço Web Access to DICOM Persistent Objects (WADO) para responder às limitações existentes em termos de distribuição e visualização de conteúdos sobre WEB [34]. Os pedidos ao servidor são efectuados sobre os protocolos Hypertext Transfer Protocol (HTTP/S), pelo que os clientes do serviço WADO não necessitam de ser compatíveis com DICOM. Os tipos de dados retornados por este tipo de serviço devem ser especificados aquando dos pedidos dos clientes como, por exemplo, DICOM, JPEG, RTF e XML [33]. Como o nome indica, o escopo do serviço WADO limita-se a objectos persistentes DICOM, como, por exemplo, imagens, vídeos e relatórios. Mais ainda, necessita do conhecimento antecipado dos identificadores dos objectos solicitados. Outra limitação deste serviço é o facto de não especificar formas de submissão de arquivos como, por exemplo, de relatórios [34].

2.3.2 PACS

Com o aumento da popularidade dos sensores digitais nas ciências médicas, a quantidade de informação gerada tem vindo a aumentar exponencialmente, estimando-se que volume de informação gerada por estes dispositivos seja da ordem dos Petabytes anuais [2]. Assim sendo, torna-se cada vez mais necessário implementar estruturas eficazes de organizar tal informação.

As infra-estruturas PACS (Picture Archiving and Communication System) têm vindo a ser a solução escolhida [4], integrando sistemas de imagem distribuídos e heterogéneos, fornecendo a possibilidade gestão de informação, organizando meios de visualização, análise, documentação de resultados de estudos clínicos e comportando mecanismos eficazes de comunicação [35].

Verificamos que as instituições de cuidados de saúde têm vindo a tornar-se fortemente ligadas e cooperantes. Um entrave à comunicação assenta no facto de um registo típico de um paciente ser constituído por campos como: primeiro nome, ultimo nome, endereço, telefone. O preenchimento, em distintas instituições, de campos similares com diferentes nomenclaturas e/ou formatos pode resultar em registos do mesmo paciente não relacionáveis. Existe ainda uma variedade de erros possíveis que contribuem para este tipo de situações: espaços adicionados, falta de espaços, caracteres inválidos, trocar o primeiro com o ultimo nome (especialmente em nomes asiáticos), troca de endereços, estado civil, etc. Têm vindo a ser desenvolvidos vários algoritmos com o intuito de resolver este tipo de situações. No entanto, o problema é que nenhum destes algoritmos inclui uma solução completa para o problema [3]. Além do mais os tradicionais sistemas comerciais PACS não são projectos rentáveis em poucos anos.

2.4 DICOOGLE

O módulo desenvolvido durante esta tese tem como objectivo suportar a comunicação de dados entre plataformas Dicoogle dispersas por diversas instituições, pelo que se torna pertinente uma descrição de tal plataforma.

Nos últimos anos tem vindo a ser desenvolvido o projecto DICOOGLE [5], uma infra-estrutura *open-source* e multi-plataforma para repositórios distribuídos de imagem médica em formato DICOM. Está a ser desenvolvida em Java e as funcionalidades DICOM são suportadas com recurso a biblioteca DCM4CHE. Trata-se de um conjunto de ferramentas úteis para desenvolvimento na área da saúde [36], usada para extrair elementos de dados de ficheiros DICOM. A plataforma Dicoogle também se baseia num indexador de domínio público (Apache Lucene) [37], o que implica que não existe nenhum servidor de base de dados pelo que não é necessário criar novos campos, novas tabelas, nem relações como acontece nos sistemas PACS tradicionais. Tipicamente, estes últimos guardam apenas um número reduzido de campos de interesse o que faz com que grandes quantidades de informação não sejam pesquisáveis [38]. Esta abordagem permite extrair, indexar e guardar todos os meta-dados detectados no cabeçalho da modalidade DICOM, incluindo atributos DICOM privados. Desta forma, baseia-se numa abordagem bastante genérica, sem necessidade de reconfigurações, permitindo reduzir os custos de instalação e manutenção.

Para permitir a comunicação dentro de uma rede hospitalar, o Dicoogle suporta ainda um modelo de transmissão ponto-a-ponto (p2p) baseada em Jgroups, ferramenta que assegura que as mensagens trocadas entre os participantes chegam ao seu destino pela ordem que foram enviadas.

Para suportar pesquisas e transferências de ficheiros, no contexto da camada *peer-to-peer*, quatro tipos de mensagens foram definidos: *Query Request*, *Query Response*, *File Request* e *File Response*. Os primeiros três tipos de mensagens são baseados em texto, seguindo uma estrutura XML, enquanto o *File Response* é do tipo de dados binários [5]. A funcionalidade destas mensagens será abordada mais à frente neste documento.

2.4.1 Jgroups

Como mencionado, a plataforma Dicoogle usa actualmente a ferramenta Jgroups [39] como meio de transporte de mensagens dentro da mesma rede hospitalar, surge assim a necessidade de fornecer uma visão geral das vantagens e limitações de tal tecnologia.

Existem dois conceitos chave em Jgroups, grupos e membros, em que cada membro faz parte de um ou mais grupos e um grupo é identificado pelo seu nome. O participante pode juntar-se a um grupo, enviar uma mensagem para todos os membros deste, ou para um determinado elemento e obter a lista dos elementos existentes no grupo.

De forma a permitir estas acções de forma confiável, distribuída e resistente a falhas, JGroups define um conjunto de protocolos e sequências de acções cujo detalhe pormenorizado foge do escopo deste documento. Ao nível da camada de transporte pode usar o protocolo UDP ou o TCP. A diferença principal em ambos é o facto de TCP garantir a correcta transmissão de dados, enquanto em UDP esse aspecto necessita ser tratado ao nível da aplicação.

Para transporte é tipicamente usado o protocolo UDP. Este é especialmente adequado pois permite enviar mensagens para múltiplos destinatários através de *multicast* (um-para-muitos). Esta característica é particularmente importante a quando da descoberta e ligação a um dado grupo, ao contrário do que ocorre com TCP no qual apenas são permitidas transmissões *unicast* (um-para-um), ou seja o envio de uma mensagem para múltiplos destinatários exige o reenvio desta para todos eles.

Enquanto, que o *multicast* em redes de acesso local é geralmente tido como uma vantagem, em redes de mais ampla dimensão o tráfego *multicast* é geralmente bloqueado pelos fornecedores de serviços de Internet (ISP), pois este tipo de informação pode gerar elevados níveis de tráfego. Levando este facto em consideração, os desenvolvedores da plataforma Jgroups na versão 2.12 (9 de Março de 2011), incluirão um protocolo *Relay*. Este pode ser configurado de tal forma que os

peers coordenadores do grupo passam a ser responsáveis por enviar as mensagens para os restantes coordenadores de outros grupos, fornecendo assim uma solução para comunicações entre grupos.

Embora esta seja uma solução natural para estender Jgroups para WAN (*Wide Area Network*), necessita de um conjunto de configurações adicionais. Cada membro do grupo pode ser em determinada altura um potencial coordenador do grupo, o que implica que é necessário estar acessível do exterior da LAN. Em termos de políticas de acesso numa instituição, esta necessidade pode ser bastante indesejável.

Outro entrave é o facto de numa ligação ser necessário que os controladores dos grupos que pretendem comunicar se conheçam mutuamente. Isso deverá ser feito através da configuração de um ficheiro tcp.xml no qual são descritas as configurações do protocolo TCP [40]. Na Figura 7 encontram-se representados os protocolos de baixo nível tradicionalmente usados Jgroups, é possível observar que enquanto a comunicação dentro de um mesmo grupo é tradicionalmente feita sobre UDP, entre múltiplos grupos esta é realizada sobre TCP.

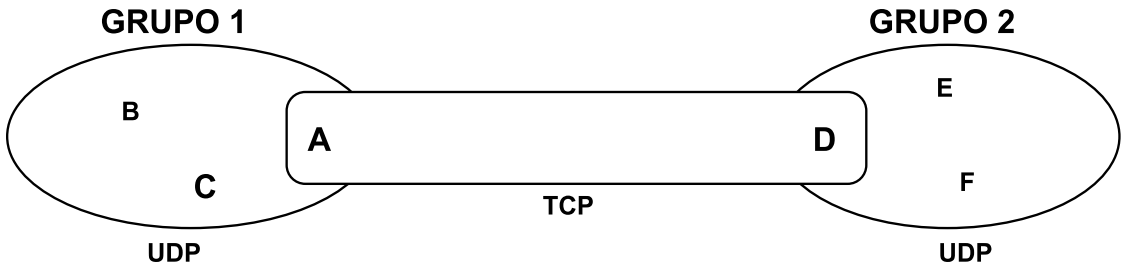

Figura 7. *Relay* Jgroups entre diferentes grupos [40]

2.4.2 Relay inicial

A quando da iniciação desta tese já se encontrava implementado, uma primeira versão do módulo de comunicações WAN baseado num serviço de *Relay* suportado pela plataforma Google App Engine e respectivos clientes. Assim, antes de descrever o trabalho efectuado, é necessário descrever previamente este módulo. O objectivo deste é garantir que as mensagens entregues em LAN através de Jgroups sejam transmitidas também em WAN, utilizando neste caso tecnologias *cloud computing*.

Como previamente enunciado, um *peer* em Jgroups pode enviar mensagens para todos os outros *peers* ou para um *peer* específico. Desta forma, na modelação do serviço de *Relay* procedeu-se a extracção dos casos de uso existentes em LAN para serem replicados em WAN. Estes encontram-se representados na Figura 8.

De seguida procede-se a uma descrição sumária destes casos de utilização.

Enviar para muitos

Finalidade	Enviar uma mensagem para todos os outros clientes.
Pré-condição	-
Parâmetros	Username do cliente, Mensagem a enviar.
Acções	Coloca a mensagem recebida no serviço Datastore.
Retorno	-

Tabela 1. Enviar mensagens para muitos peers

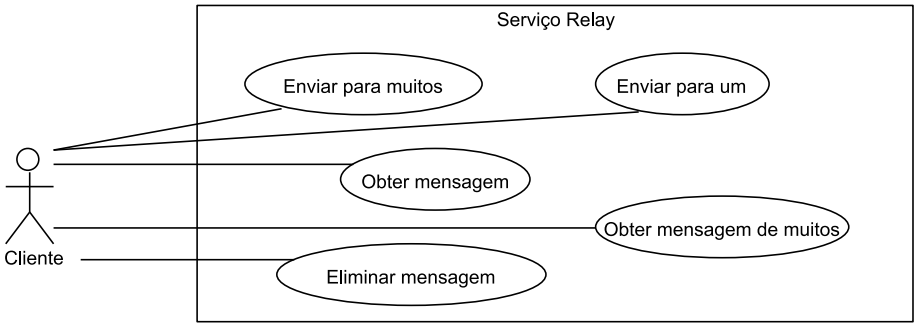

Figura 8. Casos de uso Relay inicial

Obter mensagem de muitos

Finalidade	Obter mensagens destinadas a todos os clientes existentes.
Pré-condição	É necessário que exista alguma mensagem nova no Datastore.
Parâmetros	Username do cliente, *Timestamp* da última mensagem recebida
Acções	Procura mensagens com um *Timestamp* maior que o fornecido pelo cliente. Se encontrar tal mensagem verifica se a esta não tem como origem o próprio cliente, se não for retorna-a.
Retorno	Mensagem to tipo um para muitos.

Tabela 2. Obter mensagem de muitos peers

Obter mensagem

Finalidade	Obter todas as mensagens destinadas a um dado cliente.
Pré-condição	É necessário que exista alguma mensagem cujo destinatário seja o cliente em questão.
Parâmetros	Username do cliente, threadID número que identifica *thread* do cliente que faz o pedido.
Acções	Procura por mensagens cujo destino seja o cliente especificado. Caso exista, devolve a mensagem que se encontra na posição threadID da pesquisa feita ao Datastore.
Retorno	Mensagem do tipo um-para-um, id da entidade devolvida, número de resultados devolvidos pela pesquisa feita ao Datastore.

Tabela 3. Obter mensagem

Enviar para um

Finalidade	Enviar uma mensagem para um determinado cliente.
Pré-condição	-
Parâmetros	Username do cliente origem, Username cliente destino, Mensagem a enviar.
Acções	Coloca a mensagem recebida no serviço Datastore.
Retorno	-

Tabela 4. Enviar para um peer

Eliminar mensagem

Finalidade	Apagar do serviço de *Relay* as mensagens destinadas a um dado cliente.
Pré-condição	Que exista a mensagem especificada
Parâmetros	Id da entidade a apagar
Acções	Efectua uma pesquisa no Datastore pela entidade especificada, envia um pedido para apagar todos os resultados obtidos.
Retorno	-

Tabela 5. Eliminar mensagem

O modelo de dados usado no *Relay* Inicial para suporte das acções acima mencionadas encontra-se representado na Figura 9.

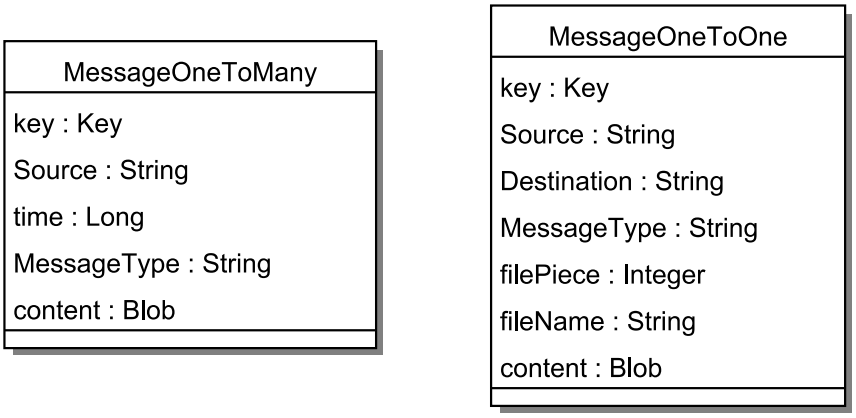

Figura 9. Modelo de dados inicial

2.4.3 Cliente inicial

Uma vez que Google App Engine não suportava mecanismos de notificação de chegada de mensagens e, de forma a resolver esta limitação, os clientes interrogavam periodicamente o serviço de *Relay* relativamente a novas mensagens através de um mecanismo de *pooling* com periodicidade de um segundo.

No caso das mensagens um-para-muitos, o cliente invocava o método correspondente ao caso de utilização *Obter um para muitos*. Se existissem mensagens novas, a mais antiga seria devolvida para que esta pudesse ser processada pelos clientes destino.

No caso de mensagens um-para-um era invocado o método corresponde ao caso de utilização *Obter mensagem*. De forma a tirar partido de *multi-threading* era devolvido o número de mensagens em espera de serem tratadas por aquele cliente. Com isso este podia lançar threads de forma a paralelizar a recepção de mensagens.

Para além do tratamento de notificações, o sistema *multi-thread* era também responsável pelo envio de mensagens. Assim, sempre que se pretendia enviar uma mensagem era verificado se tinha atingido o número máximo de mensagens a enviar em simultâneo. Caso ainda fosse possível enviar a mensagem, esta seria enviada para o serviço de *Relay* e, consoante o tipo de mensagem, era invocado o caso de utilização *Enviar para muitos* ou *Enviar para um*.

Somente no final da correcta recepção da mensagem, por parte do cliente destino, era invocado o método correspondente a *Eliminar mensagem*. De salientar ainda que, para mapear estes casos de utilização em pedidos WEB, foram criadas duas *servlets* capazes de processar mensagens HTTP dos tipos Post, Get e Delete.

Na Figura 10 encontra-se representada o funcionamento do mecanismo descrito para envio de uma mensagem do tipo um-para-muitos. É possível observar que, como existem múltiplos clientes destino, pelo que no final não é enviado um pedido para eliminar este tipo de mensagem pois não se sabe a partida quantos clientes a irão receber.

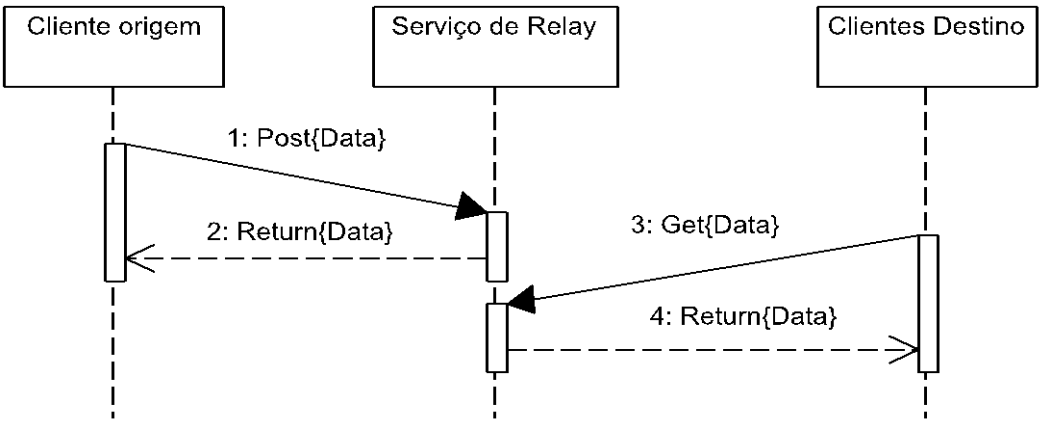

Figura 10. Fluxo mensagens inicial um-para-muitos

Na Figura 11 encontra-se representado o envio de uma mensagem do tipo um-para-um.

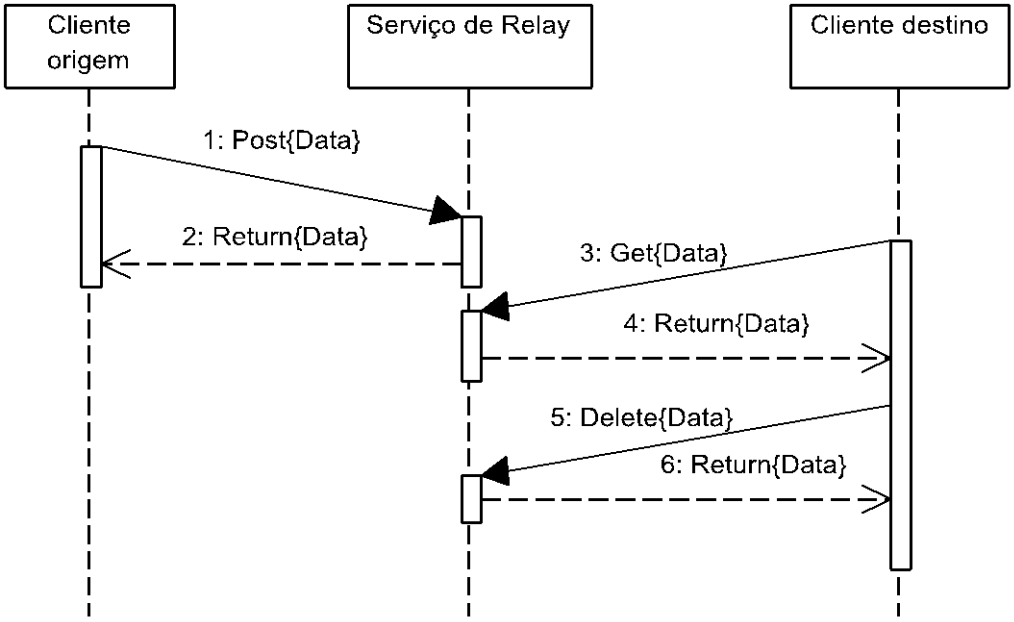

Figura 11. Fluxo mensagens inicial um-para-um

Dado o carácter inovador e pouco maturo descrito, estes mecanismos sofrem de um conjunto de problemas:

- Tendo em conta que cada pedido apenas pode demorar 30 segundos, nada garante que o armazenamento de uma dada mensagem não falhe antes ser colocada na infra-estrutura do Google App Engine.
- Cada Get e Delete correspondem a uma pesquisa sobre a base de dados de suporte à informação. O que, segundo documentação existente em Google App Engine [41] é uma operação demorada e dispendiosa em termos de recursos.
- Outro problema desta abordagem, assenta no facto de que enquanto um determinado cliente destino obtêm mensagens o cliente de origem pode continuar a colocar informação na base de dados de mensagens, o que na prática se traduz numa mudança dos resultados obtidos entre diferentes pedidos de diferentes *threads* e com isso a chegada de mensagens repetidas ao cliente destino.

- Supondo que existem mensagens destinadas a determinado cliente e este abandona o sistema antes de enviar o pedido para eliminar tais mensagens, estas mensagens permanecerão no serviço de *Relay*.

Apesar dos problemas acima mencionados, esta infra-estrutura forneceu uma importante base para o desenvolvimento do trabalho apresentado nesta tese.

2.5 Dicoogle Mobile

Existem evidências que sugerem que muitos dos erros de diagnóstico se devem a métodos ineficientes de acesso à informação, bem como a limitações temporais existentes em determinados diagnósticos como, por exemplo, o caso de urgências [42].

Quando olhamos para as localizações onde poderá ser necessário efectuar tais diagnósticos, facilmente se constatar que estes poderão necessitar de ser efectuados dentro ou fora das instituições hospitalares. Para suportar as novas necessidades de mobilidade associadas ao fluxo de trabalho de uma instituição é necessário recorrer a tecnologias móveis como, por exemplo, computadores de bolso, portáteis e telemóveis.

O uso de tais de dispositivos tem vindo a massificar-se nos últimos anos, estando presentes em praticamente todos os aspectos da sociedade actual. O que os tornam a escolha ideal, não só do ponto de vista tecnológico mas também para uso diário das equipas responsáveis por efectuar diagnóstico médico. No entanto, e para que estes dispositivos possam ser eficazes na sua tarefa de diagnóstico é necessário fornecer uma infra-estrutura de comunicação nas quais estes se encontrem devidamente integrados com o restante sistema de diagnóstico [43].

A aplicação prática de tais dispositivos no contexto médico pode ser bastante variada, podendo-se identificar alguns exemplos: monitorização de pacientes, serviços de assistência médica baseados em localização, resposta a emergências médicas e tarefas para registo de pacientes.

Desta forma, é inegável a mais-valia que este tipo de dispositivos representa, quer para as equipes médicas, quer para os seus pacientes [44]. Tendo em conta tais benefícios, e no contexto da plataforma Dicoogle, tem sido desenvolvida uma versão Dicoogle Mobile, uma aplicação móvel desenvolvida para o sistema operativo Android da Google [45].

Tal sistema operativo utiliza um kernel Linux e actualmente existem diversos dispositivos móveis que o suportam. Este sistema operativo adapta-se tanto ao formato dos tradicionais *smart phone*, bem como a dispositivos de maiores dimensões o que o torna adequado para visualização de conteúdos. Suporta também tecnologias de comunicação, tais como: Bluetooth e Wifi [44]. De forma a fornece uma conveniente base para o desenvolvimento de aplicações permite também executar aplicações Java.

A plataforma Dicoogle Mobile tem vindo a ser desenvolvida para aproveitar as potencialidades fornecidas pelos dispositivos que suportam o sistema Android, bem como a capacidade de efectuar pesquisas sobre a plataforma Dicoogle[2]. No inicio do trabalho realizado nesta dissertação esta aplicação possui-a funcionalidades que lhe permitiam efectuar pesquisas remotas por exames indexados pela plataforma Dicoogle. Para isso era usado um PacsBroker que convertia pedidos HTTP em pedidos DICOM e os encaminhava para a plataforma Dicoogle tradicional. Na resposta, o PacsBroker era responsável por traduzir as respostas DICOM para HTTP com as imagens disponibilizadas no formato JPEG, prontas a serem visualizadas em dispositivos Android.

A plataforma mobile encontrava-se assim inteiramente dependente de um sistema intermédio, i.e. o serviço PacsBroker, para se poder comunicar com o universo PACS/DICOM.

[2] Sempre que se referir apenas Dicoogle está-se a referir a plataforma mais tradicional sem qualquer tipo de tratamento de questões de mobilidade.

3 Análise de requisitos e arquitectura proposta

Neste capítulo são apresentados os requisitos do sistema desenvolvido de forma a permitir comunicações WAN. Serão ainda seleccionadas as tecnologias consideradas adequadas. Finalmente, serão descritas implementações deste sistema, quer com vista ao desenvolvimento do módulo de reencaminhamento de mensagens, quer para criação dos clientes que irão depender deste para se comunicarem entre si.

3.1 Requisitos Funcionais

Este módulo foi desenvolvido como um subsistema da plataforma Dicoogle, possui requisitos similares aos actualmente presentes no módulo para transmissão de dados sobre LAN, mas no contexto WAN. Assim, pode-se recorrer ao que já se encontra implementado em LAN, analisar os fluxos de mensagens existentes e funcionalidades permitidas pela plataforma Jgroups. Desta forma podemos chegar a uma descrição do que é necessário implementar em WAN.

3.1.1 Pesquisa

Num sistema em que se pretende que exista intervenção do utilizador é necessário fornecer alguma forma de interacção com a informação de forma eficiente. Neste contexto, o módulo WAN tem de permitir as pesquisas de dados dos pacientes e garantir que estas são recebidas pelos restantes clientes. A semelhança do que ocorre quando se efectua uma pesquisa, sobre um motor de busca, em que, tipicamente, existem múltiplas máquinas a enviar resultados para o cliente, também numa rede p2p isto acontece.

Visto que as mensagens necessárias já se encontram definidas pelo módulo LAN, o objectivo do desenvolvimento do módulo WAN é o encaminhamento das mensagens. Depois de garantir a recepção destas, pode usar-se a infra-estrutura previamente criada para processamento das mesmas. Desta forma, torna-se pertinente a descrição das mensagens previamente definidas para permitir pesquisas sobre LAN:

- ***Query Request*:** é definida sob uma estrutura XML, os seus valores são assim enviados dentro de *tags*. Uma descrição dos campos que constituem este tipo de mensagens pode ser encontrada na Tabela 6.

Parâmetro	Descrição
MessageType	É do tipo String e tem por objectivo descriminar o tipo de mensagens. No caso é preenchido com "QUERY REQUEST".
QueryNumber	Parâmetro usado de forma a ignorar resultados de pesquisas anteriores. Por exemplo, se um peer efectuar duas pesquisas seguidas, sem este campo seria ser problemático garantir que os resultados obtidos realmente interessavam aos utilizadores.
Query	Trata-se da String com a expressão de pesquisa para o motor de pesquisa Lucene, usada para que os peers possam efectuar pesquisas sobre os seus índices locais.
Extrafield	Como previamente mencionado um ficheiro DICOM pode conter uma grande quantidade de campos de interesse, pelo que enviar indiscriminadamente todas as informações existentes para as pesquisas efectuadas remotamente representaria um desperdício evitável de largura de banda e de processamento das mensagens. Assim, as mensagens *Query Request* possuem uma quantidade variável de campos extra, que permitem descriminar a informação relevante.

Tabela 6. *Query Request*

Quando um peer recebe uma destas mensagens, interroga o seu índice local e envia mensagens do tipo *Query Response*.

- ***Query Response*:** são também elas definidas em XML, contém resultados para o peer que fez uma pesquisa [5]. Na seguinte tabela são apresentados os campos que constituem estas mensagens:

Parâmetro	Descrição
MessageType	É responsável por indicar que tipo de mensagem se trata.
QueryNumber	Indica qual o número da pesquisa para o qual fornece resultados.
SearchResult	Este item contém todos os campos obrigatórios que fazem sempre parte de um resultado de uma pesquisa e são eles: Nome do ficheiro, uma sequência de controlo deste e o tamanho de mesmo. Neste item também se encontram preenchidos os campos opcionais especificados numa mensagem do tipo *Query Request*.
SearchResults	Funciona como delineador de uma sequência de itens SearchResult.

Tabela 7. *Query Response*

O fluxo de mensagens correspondente a uma pesquisa pode ser melhor entendido através do diagrama de actividades da Figura 12.

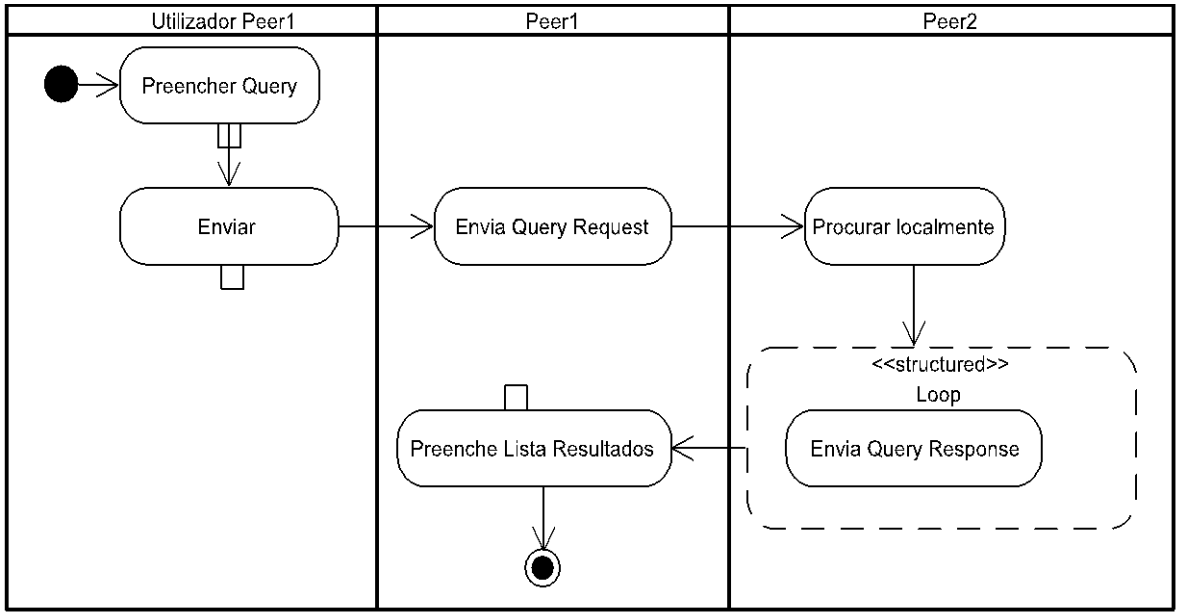

Figura 12. Procurar ficheiros

Como é possível observar pela Figura 12, um utilizador que deseje efectuar uma pesquisa começa por preencher os campos referentes a esta. Ao submete-la, a parte lógica da sua aplicação (representada na figura como Peer1) é responsável por converter esta pesquisa numa mensagem do tipo *Query Request* e então envia-la.

Um segundo peer que recebe esta mensagem, efectua uma pesquisa no seu índice local e então envia os resultados na forma de mensagens *Query Response*.

Ao receber estas mensagens o Peer1 preenche uma lista de resultados.

3.1.2 Transferência de ficheiros

O desenvolvimento deste módulo centra-se essencialmente na transmissão de imagens médicas que são armazenadas em ficheiros DICOM. O utilizador, depois de visualizar e seleccionar os resultados que respondem às pesquisas que efectuou, tem a possibilidade de solicitar

a transmissão destes para a sua máquina local. Para suportar este mecanismo foram definidos dois tipos de mensagens:

- ***File Request*:** é enviado por um peer que deseja um ficheiro. Possui os campos <FileName> e <FileHash> recebidos numa *Query Response*, os quais identificam de forma unívoca o ficheiro que se pretende.
- ***File Response*:** As imagens solicitadas são transmitidas usando mensagens *File Response*, que possuem dados binários do ficheiro a transmitir.

O fluxo de mensagens correspondente a transmissão de um ficheiro pode ser melhor entendido através do diagrama de actividades da Figura 13.

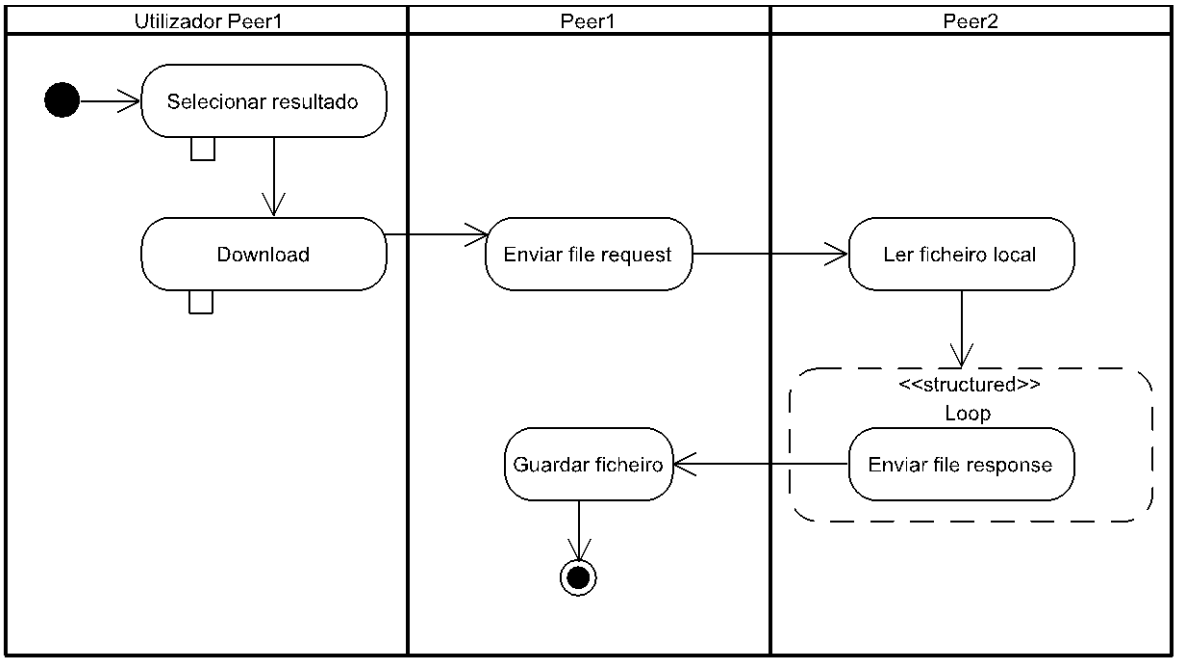

Figura 13. Descarregar ficheiro

É possível observar pelo diagrama da Figura 13, que depois de um utilizador solicitar a transferência de um ficheiro para a sua máquina local. A parte lógica da aplicação é responsável por traduzir esta solicitação numa mensagem do tipo *File Request*. A qual ao ser processada pelo peer destino o qual executará a leitura do ficheiro e consequente tradução deste em mensagens *File Response*, que serão enviadas pela rede. Estas ao serem recebidas pelo cliente que efectuo o pedido podem ser usadas para reconstituir o ficheiro.

3.1.3 Presença

Uma vez que se pretende que cada utilizador possa obter dados de múltiplos clientes, é de todo conveniente que cada cliente possua informação actualizada sobre os restantes clientes ligados ao serviço. Assim os clientes têm de ser notificados de quando existem mudanças de estado dos restantes clientes. Por exemplo, quando um novo cliente se liga ao sistema ou se desliga.

3.2 Requisitos não funcionais

O módulo desenvolvido no contexto deste trabalho tem como objectivo a partilha de imagem médica entre dispositivos dispersos por uma rede de acesso global (WAN). Dada a heterogeneidade de utilizadores existentes numa rede desta natureza e o carácter confidencial da informação a transmitir, bem como a rápida evolução das tecnologias envolvidas, foi necessário desenvolver este módulo tendo em conta um conjunto de requisitos não funcionais:

- **Fácil de utilizar:** Os serviços disponibilizados devem necessitar do menor esforço possível por parte do utilizador final, sendo assim necessárias interfaces intuitivas.
- **Estrutura Simplificada:** de forma a garantir que os custos de manutenção de uma infra-estrutura desta natureza sejam baixos, é necessário que a infra-estrutura seja simples.
- **Seguro:** dado o carácter confidencial da informação a transmitir é necessário garantir a privacidade bem como a integridade desta.
- **Estável:** um serviço desta natureza deve ser estável. Por exemplo, a falha num determinado cliente não deve colocar em questão o correcto funcionamento dos restantes [46].
- **Escolha de protocolos:** geralmente os acessos ao exterior da instituição são controlados por *firewalls* que restringem os protocolos bem como as aplicações que podem ser usadas para comunicar com o exterior. Por um lado, estas *firewalls* são especialmente importantes pois reduzem os riscos existentes com comunicações para o exterior da infra-estrutura. Por outro lado, podem colocar políticas demasiado restritivas que dificultem as comunicações entre instituições [7]. Desta forma, com o objectivo de maximizar as probabilidades dos clientes que usem este módulo passem por estas *firewalls*, sem necessidade de configurações adicionais, são usados protocolos de comunicação geralmente permitidos para transmissão de conteúdos. Um protocolo que é um bom candidato para este propósito é o HTTP pois geralmente é usado para transportar conteúdos Web tais como: páginas, e transferência de ficheiros.
- **Necessidade de servir pedidos externos:** actualmente é natural que o número de endereços disponíveis para comunicações externas a infra-estrutura se encontre limitado a um pequeno conjunto, muito menor que o número de máquinas existentes em LAN. Quando se efectua um pedido ao exterior, a gateway responsável pela retransmissão troca geralmente o endereço origem pelo endereço da sua interface externa, de forma que a resposta seja enviada para esta interface. Quando recebe a resposta encaminha-a para o elemento da LAN que fez o pedido. Este mecanismo, denominado *Network address translation* (NAT), permite que se encontrem ligados muito mais computadores à Internet que a gama de endereços possível com endereços IPv4. O problema é que para que um nó seja capaz de receber pedidos externos é necessário configurar o elemento responsável pelo acesso ao exterior. Uma forma de contornar este problema é o uso de IPv6 em que a gama de endereços é suficientemente grande para não necessitar de tal mecanismo. No entanto, nem sempre o IPv6 encontra configurado.
- **Descoberta:** uma questão pertinente é "como um dado nó sabe para onde deverá enviar as suas mensagens?". Por exemplo, como é efectuado o processo de descoberta dos restantes peers.

3.3 Análise de requisitos

Uma possível resposta para a questão levantada no requisito **descoberta**, pode ser a existência um sistema centralista, no qual todos os clientes que desejam comunicar reportam eventos a esta mesma entidade. Assim, fica resolvido o problema de descoberta de clientes existentes. O problema desta solução é que uma eventual falha da unidade central, o sistema como um todo inevitavelmente falhará, o que é algo contra o requisito considerado de **estabilidade.** A solução considerada para garantir a estabilidade do sistema consistiu em colocar esse sistema central em algum fornecedor de serviços *cloud*. Ou seja, se o próprio serviço visto como centralista for ele próprio distribuído, será possível atingir o mesmo nível de confiança no sistema final, que seria espectável com uma solução com ligações ponto-a-ponto. Um bom exemplo de um serviço que fornece tal grau de confiança é o motor de pesquisa Google, o qual se falhar comummente o utilizador pensará que a ligação a Internet falhou [6].

Uma vez, que devido a configurações de NAT e de *firewall*, podem não ser permitidas ligações ponto-a-ponto entre os diferentes clientes sem parametrizações adicionais, este sistema centralista deve também fazer o reencaminhamento das mensagens entre clientes. Assim, uma escolha importante desta dissertação foi qual dos serviços *cloud* oferece a melhor proposta tendo em conta os requisitos existentes.

Salesforce foi descartado à partida pois não possuía um serviço suficientemente flexível para consistir só por si uma opção viável. Uma vez que qualquer um dos serviços anteriormente considerados possui uma boa reputação, foi considerado que qualquer um deles permite obter um serviço com o mesmo grau de estabilidade e segurança. Em termos de escolha de protocolos, qualquer um dos serviços suporta transmissão HTTP. Google App Engine foi considerado como o serviço que oferece a melhor alternativa pois o balanceamento dos pedidos é gerido pela própria plataforma *cloud* o que não ocorre nas restantes plataformas consideradas. A sua estrutura permite que um serviço cresça consoante as suas necessidades sem precisar de reconfigurações adicionais, permitindo assim obter uma estrutura simplificada e resistente a falhas. Além do mais, o serviço de *Relay* retratado no segundo capítulo encontrava-se implementado neste serviço, o que nos forneceu uma conveniente base tecnológica e uma referência para os testes que viriam a ser realizados. Além disso, permite usar os seus serviços de forma fácil e inicialmente gratuita, o que a torna uma plataforma ideal para desenvolvimento e divulgação do serviço que se pretende desenvolver.

3.4 Arquitectura do módulo WAN

Tendo em conta os requisitos existentes e a complexidade das plataformas envolvidas optou-se por uma abordagem iterativa com aproximações sucessivas à solução do problema. Nesta secção serão descritas as iterações nas quais este módulo foi desenvolvido.

3.4.1 Primeira Iteração

Como previamente mencionado, já se encontrava em desenvolvimento um módulo para transmissões WAN sobre Google App Engine. Desta forma, o trabalho desenvolvido focou-se em melhorar o serviço previamente existente.

Assim inicialmente foi definido um mecanismo de *login* pois, dado o carácter confidencial da informação a enviar, é conveniente definir que clientes se podem ligar. Para isso foram implementados métodos de autenticação baseados em *username* e *password*, ou seja, o cliente para se ligar ao serviço de *Relay* tem de fornecer as suas credenciais, sendo estas verificadas. Na Figura 14 estão representadas as classes para suporte desta solução. Como é possível observar ambas as entidades apontam-se entre si, sendo possível obter uma através da outra.

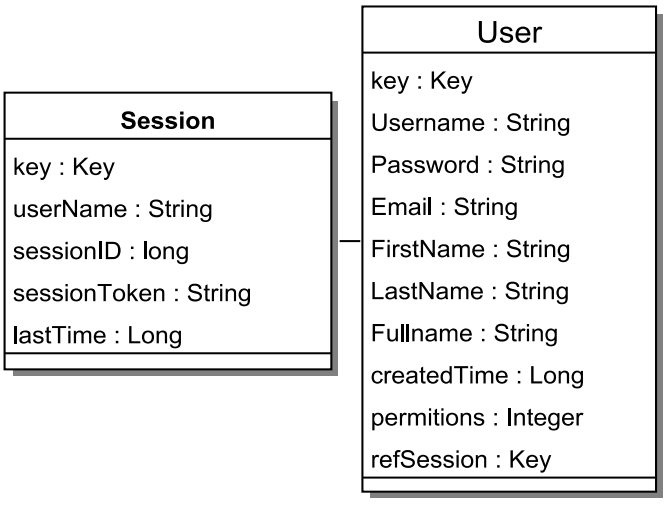

Figura 14. Diagrama classes para suporte ao conceito sessão

Nas entidades presentes na Figura 14 os atributos das entidades *User* possuem objectivos facilmente reconhecidos pelos seus nomes atributos nas entidades *Session* requerem mais uma descrição, a qual está presente na Tabela 8.

Parâmetro	Descrição
Key	Identifica de forma unívoca uma entidade do tipo *Session*.
Username	Nome do utilizador a quem pertence a sessão.
sessionID	Implementa a conceito de propriedade de mensagens. Uma vez que este atributo é do tipo long, permite que todos os dados referentes a um dado cliente referiram a sessão a qual pertencem. No momento de *logout* é então possível pesquisar dados referentes a uma da sessão e proceder a respectiva eliminação destes.
sessionToken	Este parâmetro é retornado pelo *Relay* no processo de *login* e identifica de forma única a sessão do cliente. É passado em todos pedidos seguinte feitos pelo cliente.
lastTime	Hora do último pedido feito pelo cliente, usado para determinar os clientes activos.

Tabela 8. Descrição classe sessão

De forma a optimizar o serviço previamente existente a primeira abordagem ao problema partiu de dois pressupostos:

- Pesquisas sobre o serviço GAE devem ser reduzidas tanto quanto possível de forma a evitar gastos desnecessários, reduzindo assim o tempo de permanência dos dados no serviço de *Relay*.
- Mensagens repetidas transmitidas sobre WAN contribuem de forma bastante negativa para o desempenho global do sistema.

A solução escolhida foi optimizar os mecanismos referentes as mensagens um-para-um[3], pois estas representam a maior parte do volume de informação transferido. Para isso, começou-se por dividir o processo de obtenção de dados em três fases:

- Numa primeira fase é feita uma pesquisa sobre o serviço Datastore por mensagens que pertençam a um determinado cliente. Os resultados passam a ser apontados por um conjunto de metadados, que possuem uma chave sequencial para que possam ser obtidos pelo cliente de forma explícita e sem necessidade de pesquisas adicionais.
- Numa segunda fase os clientes especificam exactamente qual o número da mensagem que pretendem obter.
- No final de todas as mensagens da primeira fase terem sido transmitidas para o cliente destino é enviado um pedido para apagar os dados destas.

Na Figura 15 está representado o modelo de dados para suporte a esta solução. Como é possível observar, o armazenamento das mensagens um-para-um, que se pretende retransmitir, encontra-se separada em diferentes tipos de entidades com diferentes propósitos:

MessageOneToOneMetadata: Representa os dados associados ao módulo WAN que permitem identificar as características dos diferentes conteúdos das mensagens a enviar. Por exemplo, qual dos quatro tipos de mensagens se trata.

MessageOneToOneContent: Representa o conteúdo das mensagens.

[3] Como pode ser observado na subsecção 2.4.2 na arquitectura inicial do serviço de *Relay* foram definidos dois tipos de mensagens:

- um-para-um: Destinam-se a um determinado cliente.
- um-para-muitos: Destinam-se a todos os clientes.

MessageOneToOneSearchable: Aponta para mensagens dos dois tipos *Metadata* e *Content*. Estão separadas para reduzir a quantidade de dados retornados na primeira fase do processo.

MessageOneToOneFetch: Possui as chaves sequenciais geradas na primeira fase de obtenção de dados e apontam para os restantes tipos de mensagens.

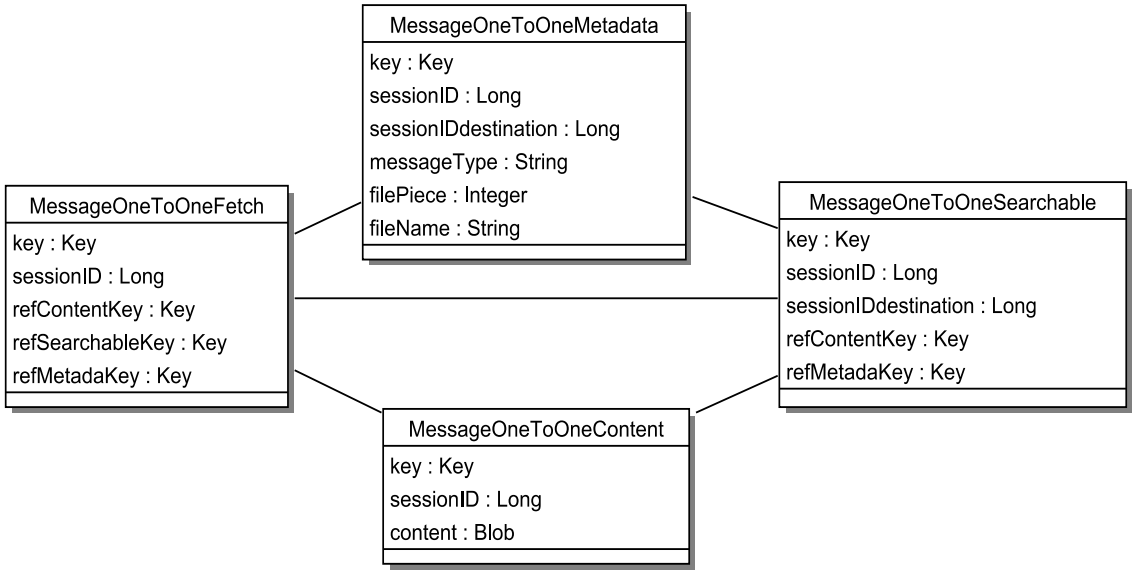

Figura 15. Modelo de dados *Polling*

Na Figura 16 encontra-se representado o fluxo de mensagens responsável por efectuar as acções acima descritas. Como é possível constatar, na segunda fase do processo é possível beneficiar de paralelismo na leitura das mensagens.

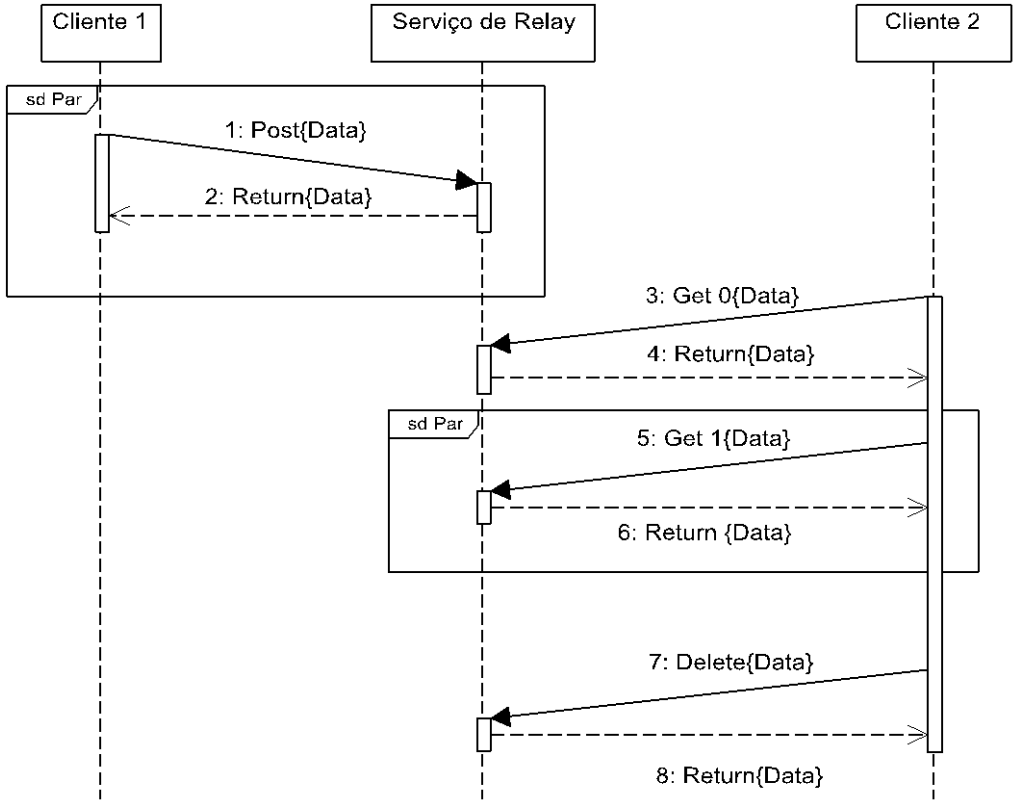

Figura 16. Fluxo de mensagens *Polling*

Uma observação pertinente na análise da Figura 16 é que apesar dos pedidos dos dados das mensagens propriamente ditos serem efectuados de forma paralela, a primeira e da última fase do processo de obtenção de mensagens para cliente 2 (Destino) são efectuados de forma sequencial, sem beneficiarem de qualquer paralelismo. Esta questão foi resolvida com recurso a reestruturação dos identificadores[4] das entidades MessageOneToOneFetch existentes, ou seja a criação de metadados para leitura de entidades funciona como se trata-se de duas filas, alternando de uma fila para outra e promovendo desta forma o paralelismo das operações de leitura.

Como previamente mencionado, cada cliente interroga periodicamente o serviço de *Relay*. Este facto foi aproveitado para apagar as mensagens de um determinado cliente. Assim, de 90 em 90 segundos é lançado um evento. Neste, é feita uma pesquisa sobre o Datastore por todos os clientes que não realizem *polling* há mais de 90 segundos e, se existirem tais clientes, é efectuado o *logout* destes.

Como referido, o serviço GAE disponibiliza um serviço denominado Memcache. Na prática trata-se de um serviço que permite a uma aplicação armazenar e ler dados de uma memória distribuída e é bem mais rápido que o serviço Datastore. No entanto, a quantidade de dados que é possível armazenar em Memcache é bastante limitada. Esta limitação foi de facto testada, procedendo-se ao armazenamento de uma determinada quantidade de dados neste serviço e consequente tentativa de acesso a estes dados. Foram testadas várias quantidades diferentes de dados, chegando-se ao valor de 150 MB como sendo o limite máximo de dados que podem ser armazenados neste serviço sem consequente perda de informação. Assim, não a consideramos uma alternativa fiável para armazenamento temporário de mensagens. Contudo, em mecanismos em que se proceda repetidamente a leitura dos mesmos valores, constitui uma possibilidade interessante para poupar recursos e melhorar o desempenho da solução.

Desta forma, o mecanismo anteriormente descrito para verificação de mensagens um-para-muitos foi optimizado fazendo uso do serviço Memcache, no qual é gravada a hora da última mensagem enviada para o *Relay* e caso o cliente tenha um valor mais antigo, é retornada uma nova mensagem. Se a leitura da hora da última mensagem falhar, é então feita a pesquisa da última mensagem existente no serviço Datastore. Desta forma é usado o serviço Memcache para optimizar pesquisas sem se comprometer a fiabilidade do sistema.

Como já referido, de forma a limitar o acesso ao serviço de *Relay* apenas aos clientes deste, é necessário fornecer um *sessionToken* em todos os pedidos efectuados. Desta forma, e para optimizar o processo de verificação desta condição, as entidades *Session* são armazenadas na Memcache, conseguindo-se optimizar a verificação de *tokens* do cliente.

Com o objectivo de evitar erros com armazenamento de dados e possíveis mensagens perdidas, foram também implementados mecanismos que, em caso de erros no Datastore, esperam um tempo aleatório e voltam a tentar a escrita de dados, em último recurso devolvem um código de erro ao cliente que efectuou o pedido.

De forma a optimizar o processo de transmissão, foi feito um teste que constituiu em verificar como o tamanho das mensagens influenciam o desempenho do sistema. Para isso foram então medidos os tempos das transferências, de forma a estudar o compromisso que existe entre paralelismo do envio de mensagens e o tamanho destas:

- Reduzindo o tamanho das mensagens é possível aumentar o paralelismo de transmissão destas a custo de mais espaço despendido para cabeçalhos HTTP e necessidade de iniciar o processo de transmissão de mensagens mais vezes.
- Aumentando o tamanho das mensagens é possível reduzir o número de ligações abertas para a transmissão da mesma quantidade de informação mas reduzindo o paralelismo com que as mensagens são enviadas.

Na Figura 17 é apresentada qual a evolução do tempo de transmissão de um ficheiro de 18 MB sobre o módulo WAN em relação a variação do tamanho considerado para cada mensagem.

[4] Em Datastore GAE os identificadores de entidades são strings no caso específico como o formato *sessionID + "T" + numPointerFetchQueueNumber + "T" + i*.

Com base nos resultados obtidos na Figura 17, o tamanho por defeito de cada mensagem foi definido para 819 KB, uma vez que este valor pertence a zona onde existiu melhor desempenho.

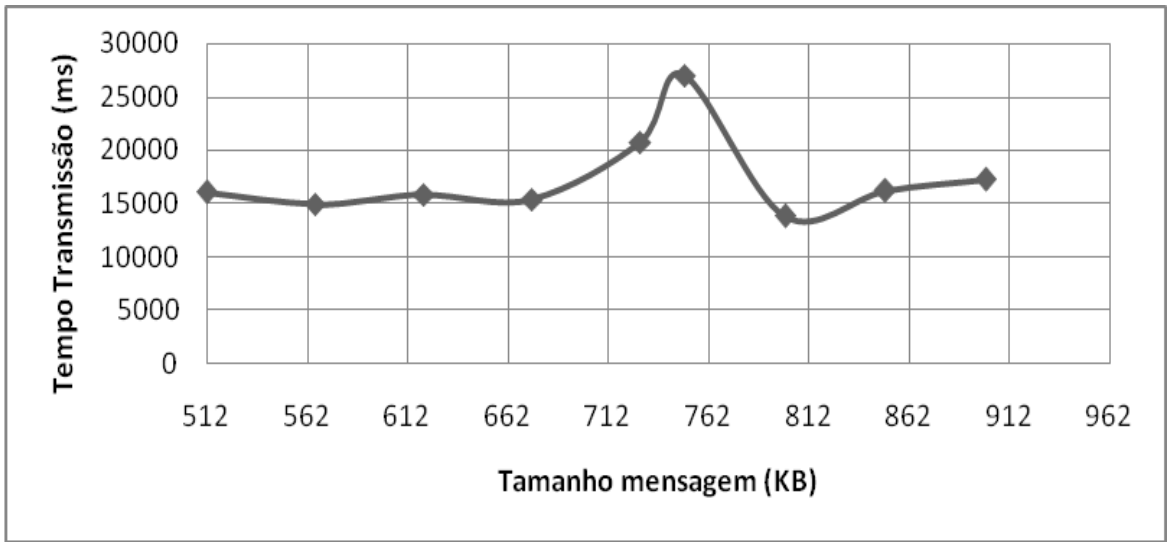

Figura 17. Relação tamanho mensagens com tempo transmissão ficheiro 18MB

3.4.2 Segunda Iteração

Na primeira abordagem para o desenvolvimento do serviço de *Relay* é possível salientar alguns problemas:

- De forma a garantir que as mensagens demorem pouco tempo a chegar ao seu destino é usado um mecanismo de *pooling* activo com uma periodicidade bastante baixa. O que pode representar um desperdício de largura de banda com eventuais pedidos sem quaisquer resultados. Ou seja, o serviço de *Relay* pode ser sujeito a uma certa carga evitável, podendo levar a certas limitações na escalabilidade de todo o sistema [47].
- O envio de mensagens do lado do *Relay*, transformou-se numa operação relativamente complexa, tendo em conta que os serviços *cloud* se baseiam num modelo por utilização de serviços, o que representa inevitavelmente despender recursos extra para a criação de metadados para suporte da solução.
- Esta solução foi optimizada para situações em que existem muitas mensagens a chegar para o mesmo cliente e em que é possível beneficiar do paralelismo da leitura destas no serviço de *Relay*. Em situações em que não é possível beneficiar de tal paralelismo, representa uma adição do esforço para criação de metadados, sem qualquer benefício. Por exemplo, esta situação pode ocorrer se a velocidade a que as mensagens são enviadas para o serviço *Relay* for inferior à velocidade de leitura.

Com a versão 1.4.0 (lançada em Dezembro de 2010) do GAE, um serviço para transmissão de mensagens de uma aplicação App Engine para os seus clientes passou a fazer parte dos serviços disponíveis, permitindo um mecanismo de *publish-subscribe*[5]. Este serviço é denominado como Channel e usa a infra-estrutura do serviço Google Gtalk para enviar mensagens. Uma limitação do serviço Channel é que para os clientes apenas foi disponibilizada uma API Javascript [48]. Uma vez que a plataforma Dicoogle se encontra desenvolvida em Java, foi considerada a possibilidade de estudar a comunicação entre GAE, GTalk e Cliente. Com isso, pretendeu-se criar um cliente

[5] Em serviços *publish-subscribe* os clientes subscrevem um dado serviço e sempre que ocorram mudanças são alertados acerca delas.

Java para GAE Channel. O problema assenta na forma como são calculados um conjunto de números de controlo entre o GTalk e os clientes Javascript e a falta de documentação para criar tal solução.

Actualmente, existem diversas ferramentas que permitem uma aplicação Java executar Javascript. Numa primeira abordagem foram usados os métodos padrão JDK 6, mas infelizmente, a API definida pelo Channel não é suportada. A opção escolhida foi a ferramenta HtmlUnit [49], um *browser* embutido, desenvolvido em Java, multi-plataforma para teste de páginas Web, com suporte de Javascript, permitindo assim usar o GAE Channel.

Este novo recurso permite mensagens até 32K, segundo a documentação GAE, e devem ser formatadas seguindo uma estrutura *JavaScript Object Notation* (JSON) [48]. Nesse formato a informação é transmitida sob a forma de texto, pelo que, para enviar uma mensagem *File Response* através de mensagens *channel*, seria necessário converter dados binários para base 64 o que representaria um aumento do volume de dados a enviar. Como se pretende um serviço eficiente, não seria, portanto, uma boa solução. Assim, optou-se por uma estratégia no qual mensagens *channel* apenas servem para notificar os clientes da chegada de mensagens a si destinadas.

Uma vez que se pretendia estudar como a notificação da chegada de mensagens poderia influenciar o desempenho do sistema, foi necessário redefinir as interfaces de acesso ao serviço de *Relay*, considerando-se então pertinente uma análise de possíveis formas de o poder fazer.

Até ao momento, eram usados cabeçalhos HTTP nos quais eram passados os parâmetros dos diferentes casos de utilização. Os valores destes cabeçalhos tinham de ser explicitamente preenchidos pelos clientes e programados a baixo nível. Esta solução é bastante ineficiente em termos de manutenção futura do serviço. Supondo que se altera um dos cabeçalhos, então tem de se alterar a solução em dois sítios, no *Relay* e nos Clientes. De forma a minimizar esta situação foram consideradas interfaces SOAP e *Representational State Transfer* (REST):

- Os *Web Services* SOAP: descrevem de forma uniformizada os recursos disponibilizados por um serviço, e podem ser facilmente consumidos por diferentes tecnologias, como por exemplo, .NET e Java. Infelizmente para alcançar tal grau de interoperabilidade os pedidos SOAP são constituídos por um conjunto de *tags* XML, que definem um conjunto de regras de como processar o pedido, aumentando de forma indesejável o tamanho deste.
- Os *WEB Services* REST: têm vindo a ser usados para comunicações com um elevado grau de desempenho. Neste tipo de interfaces as funcionalidades do serviço são mapeadas em mensagens HTTP, por exemplo, GET, PUT, POST e DELETE. Com o objectivo de definir as operações, executáveis sobre objectos remotos para diferentes objectos, são geralmente usados diferentes localizações Web (URL) [50].

Devido ao desempenho que é possível alcançar com interfaces REST, esta foi a solução escolhida para implementar a interface do serviço de *Relay*.

De forma a tentar tirar partido do desempenho das interfaces REST, é de todo conveniente garantir métodos de tratamento de mensagens eficientes quando estas se encontram no serviço de *Relay*. Assim, foi implementado o conceito de comunidades. Estas possibilitam que as mensagens sejam enviadas de forma eficiente, pois possibilitam reduzir a quantidade de mensagens enviadas, uma vez que é possível restringir o âmbito das pesquisas efectuadas. Também permitem aos utilizadores organizarem-se consoante os seus grupos de interesse. Assim, um cliente pode enviar mensagens para todos os elementos da sua comunidade ou apenas para um deles. Podem existir dois utilizadores como o mesmo nome no serviço de Relay desde que se liguem a comunidades distintas. Assim, o método de *login* com credenciais (*username*, *password*) foi alterado para usar credenciais (*username*, chave da comunidade).

Caso não seja fornecida a chave da comunidade será verificado se está activa uma comunidade pública e o utilizador é associado a essa comunidade. Na Figura 18 encontram-se representadas duas comunidades, nas quais existe sobreposição dos usernames dos utilizadores mas estes são diferenciados pelas suas comunidades.

Descritas algumas das tecnologias escolhidas para implementar o serviço de *Relay* com uma estratégia *Publish-Subscribe*, vamos então fornecer uma descrição das funcionalidades suportadas por este.

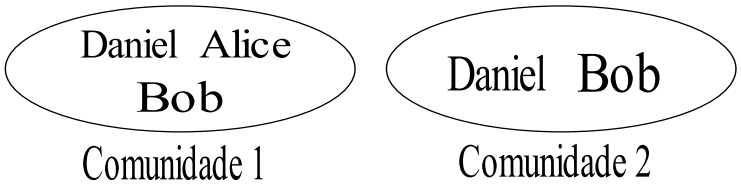

Figura 18. Elementos de duas comunidades

Nesta iteração, procedeu-se a toda uma reestruturação dos casos de uso existentes, como se encontra representado na Figura 19. Mais ainda, é possível observar que existe um agrupamento das funcionalidades disponibilizadas em diferentes pacotes, consoante a finalidade do caso de uso em questão.

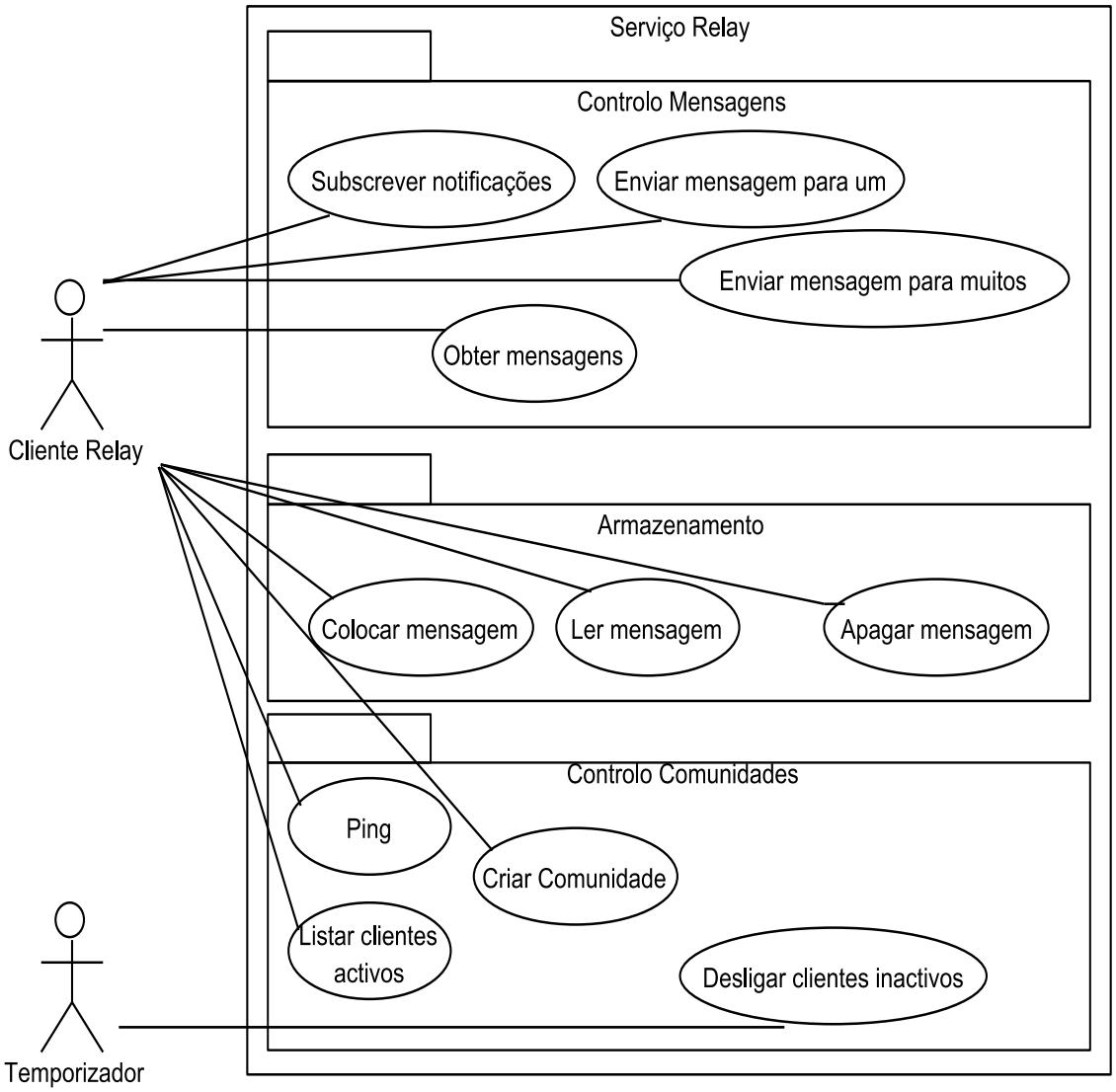

Figura 19. Casos de uso *Relay*

De seguida procede-se a uma descrição sumária das funcionalidades dos casos de utilização suportados pelo *Relay*.

Caso De Utilização	Finalidade
Criar Comunidade	Criar uma comunidade.
Listar clientes activos	Obter a lista de clientes activos na comunidade ao qual o cliente pertence.
Ping	Notificar o serviço de *Relay* que o cliente se encontra activo.
Subscrever notificações	Permitir a um cliente subscrever ou renovar o pedido para receber notificações a si destinados.
Obter mensagens	Permite a um cliente obter as mensagens de controlo a si destinadas através de polling.
Desligar clientes inactivos	Actualizar a lista de clientes com ligação activa e desligar clientes inactivos.
Enviar mensagem para muitos	Enviar uma mensagem JSON de um cliente para todos os outros existentes na sua comunidade.
Enviar mensagem para um	Enviar uma mensagem JSON de um cliente para outro.
Ler mensagem	Obter os dados de uma mensagem colocada no serviço de *Relay* por outro cliente.
Colocar mensagem	Colocar um ficheiro no serviço *Relay* acessível a outro cliente.
Apagar mensagem	Apagar os dados de uma mensagem previamente colocada no serviço de *Relay*.

Tabela 9. Casos de uso *Relay* Segunda Iteração

Até ao momento a descrição do trabalho elaborado focou-se nos casos de uso do serviço de *Relay*. Neste ponto, é pertinente descrever como foi implementado o módulo cliente que interage com o serviço de *Relay*. Para início de descrição, é apresentado no diagrama de sequência da Figura 20 o envio de uma mensagem do tipo *Query Request*. Nesta figura, escolheu-se representar os pedidos ao serviço de *Relay* como chamadas a métodos remotos.

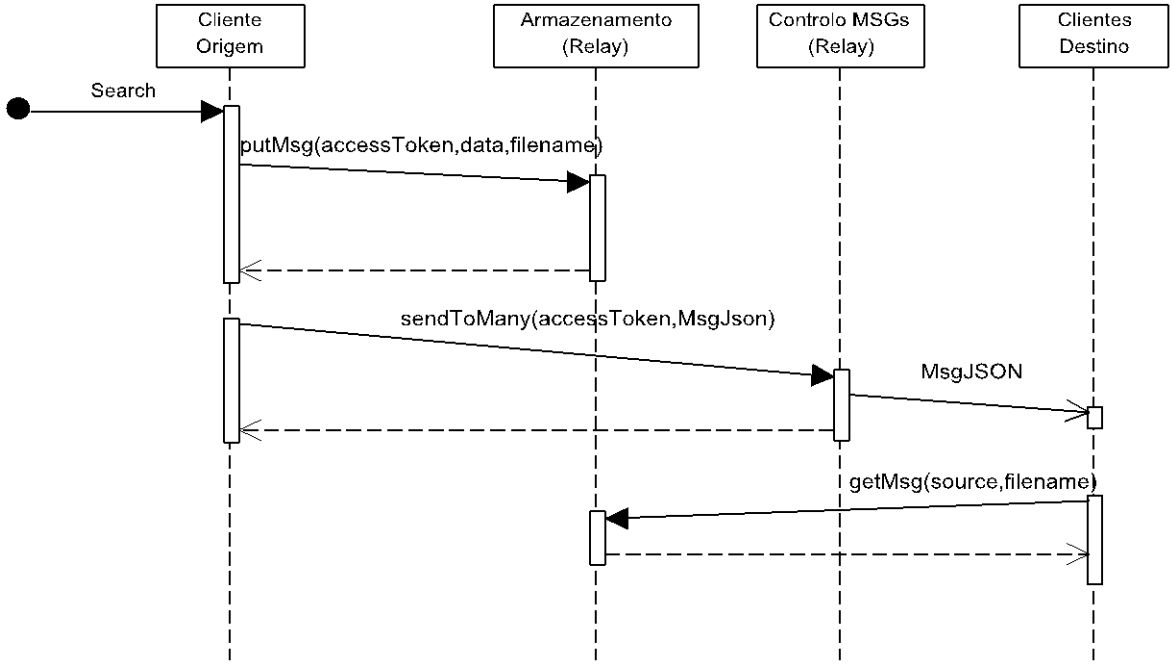

Figura 20. Sub-conjunto mensagens *Query Request*

Como é possível observar pela Figura 20, os dados de uma mensagem *Query Request* são inicialmente enviados e armazenados no serviço de *Relay*, só então os clientes destino são notificados acerca destas. São estes que fazem o pedido para leitura dos dados das mensagens. Esta solução foi escolhida com vista a maximizar a quantidade de clientes a que as mensagens *Query*

Request podem chegar. Tendo em conta que o tempo por pedido se encontra limitado a 30 segundos do lado *Relay*, enviar a mensagem de uma vez só e deixar que seja o *Relay* a distribuir pelos clientes da comunidade poderia limitar bastante a escalabilidade do sistema. Outra estratégia possível poderia ser, o cliente a enviar para cada um dos outros *peers* a mensagem. No entanto, essa solução contudo poderia aumentar significativamente a quantidade de tráfego gerado.

Outra questão que deve ser respondida é: "qual o tempo útil que uma mensagem *Query Request* deve possuir?" Nesta aproximação foi assumido que cada cliente apenas efectua um pedido de cada vez, pelo que no serviço de armazenamento é gravado sempre o ficheiro com o nome "*Query.0*". Este ficheiro apenas será apagado do serviço de armazenamento aquando da chamada do método de *logout* no qual são eliminados todos os ficheiros armazenados pelo módulo cliente, ou será reescrita quando o cliente efectuar novos pedidos.

As restantes mensagens *Query Response, File Request, File Response* correspondem a um conjunto de pedidos ao serviço de *Relay* ligeiramente diferente. No diagrama da Figura 21 estão representados os pedidos efectuados para estes tipos de mensagens.

Como se pode observar pela Figura 21 a grande diferença no envio deste tipo de mensagem, quando comparado com as *Query Request*, é o facto de no final de cada transferência ser enviado um pedido para apagar a mensagem enviada. Uma questão que pode surgir na análise deste diagrama é: Porque é que o cliente destino não apaga directamente a mensagem recebida? Neste ponto do design foi considerado o conceito de propriedade da mensagem. E foi considerado como o mais correcto assumir que as mensagens pertencem ao cliente origem uma vez é este que as envia para a rede e estas são assim da sua responsabilidade.

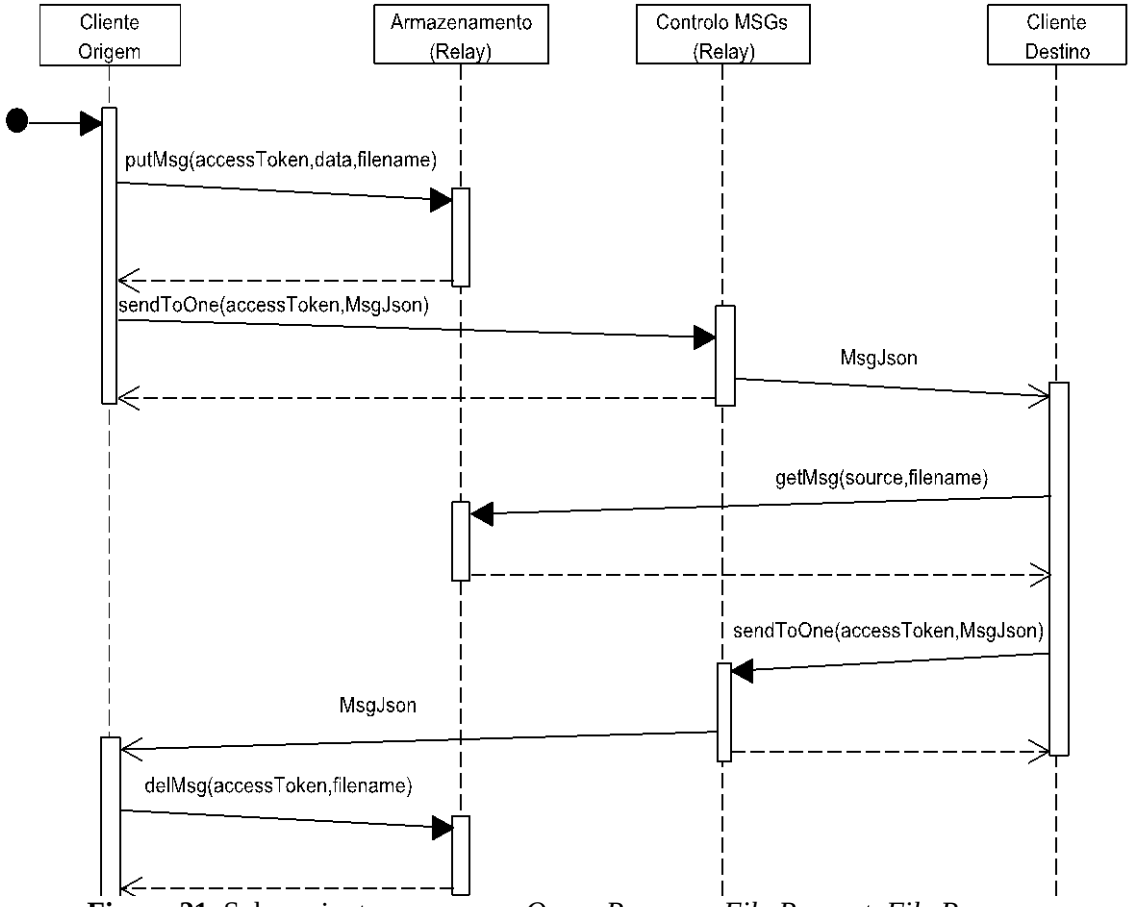

Figura 21. Sub-conjunto mensagens *Query Response,File Request, File Response*

Assim foi necessário definir um novo tipo de mensagem que pode ser recebida por um cliente, denominada *DeleteEntry*. Na Figura 22 encontra-se representada uma mensagem deste tipo.

```
{ "fromUser":"jose",
"toUser":"daniel",
"messageType":"DeleteEntry",
"content":"eyJwdWJsaWNMaW5rIjoiaHR0***dG9yYWdlTmFtZSI6InJlbGF5R0FFIn0\u003d"
}
```


Figura 22. Mensagem *DeleteEntry*

Esta mensagem é constituída pelos campos descritos na tabela abaixo:

Parâmetro	Descrição
messageType	Representa o tipo da mensagem.
fromUser	Especifica o cliente do qual a mensagem foi enviada.
toUser	Utilizador a quem é destinada a mensagem.
content	Contem informações codificadas em base 64 da localização do fragmento a apagar.

Tabela 10. Mensagem *DeleteEntry*

Ao contrário do que ocorre com uma mensagem do tipo *Query Request*, podem existir múltiplas mensagens enviadas em simultâneo pelo cliente origem, pelo que os nomes dos ficheiros associados a estas mensagens são formados por "Tipo De Mensage.Id". Caso o cliente origem deixe a rede, vão deixar de existir invocações do método *ping* pelo que o serviço de *Relay* se encarregará de invocar o método de *logout* deste no qual as mensagens associadas a este cliente serão apagadas. De forma a efectuar esta monitorização das mensagens dos clientes foi necessário recorrer ao conceito de presença.

Este é outro importante conceito do módulo, isto é um determinado cliente deve possuir uma lista actualizada que identifica quais dos outros clientes que se encontram activos. Desta forma, sempre que um cliente se liga ou desliga de uma determinada comunidade, os restantes elementos são informados que ocorreu uma mudança nos elementos que constituem tal comunidade. Estes, por usa vez, efectuam um pedido ao serviço de *Relay*, o qual devolve a lista actualizada dos utilizadores ligados na comunidade. A sequência de acções que constitui o mecanismo de presença encontra-se retratada na Figura 23.

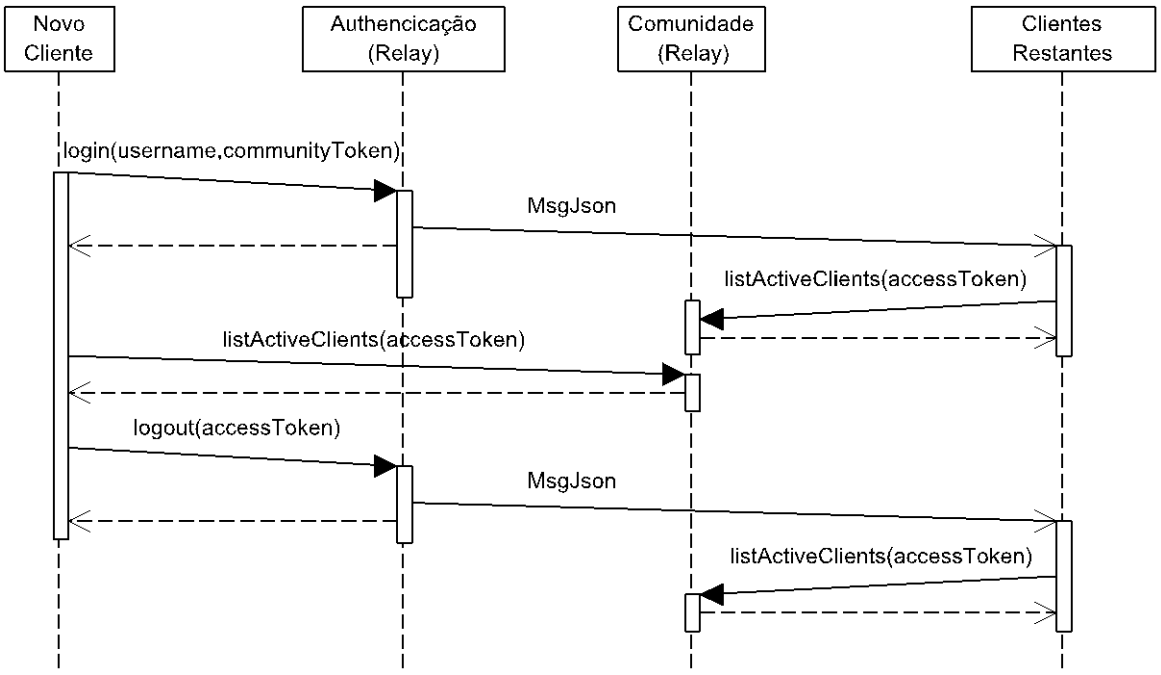

Figura 23. Fluxo de mensagens para mecanismo de presença

Como é possível observar, os restantes clientes são todos eles notificados a cerca de novos clientes da comunidade e abandonos desta. Para isso, foram usados os métodos de suporte às mensagens um-para-muitos e foi especificado um novo tipo de mensagens denominado como *ChangePeersList* que se destina a alertar os clientes para mudanças ocorridas nos estados de outros clientes. A mensagem da Figura 24 foi capturada depois de um novo cliente se ter ligado no serviço de *Relay*.

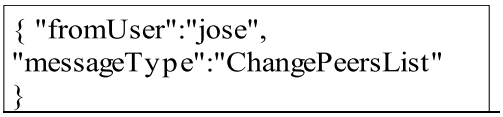

Figura 24. Mensagem *ChangePeersList*

É possível observar na Figura 23 que quando é explícito o *logout* existe uma actualização quase imediata dos clientes existentes por parte dos restantes. Os problemas surgem quando o serviço é abandonado e não é feito o *logout* explícito por parte do cliente. Para isso, foram desenvolvidos um conjunto de métodos, um dos quais é invocado com periodicidade de 120 segundos que verifica que clientes não efectuaram notificações ao sistema a mais de 90 segundos e, caso existam tais clientes, é invocado os métodos de logout destes. É então da responsabilidade do cliente notificar o serviço de *Relay* que se encontra activo e, para isso, invoca o método ping com periodicidade de 60 segundos.

3.4.3 Terceira Iteração

Como é possível constatar pela descrição feita, todo o serviço encontrava-se dependente de uma única plataforma *cloud*. Se, por um lado, isto torna a estrutura bastante simples, por outro, pode tornar os clientes demasiado dependentes de um único fornecedor de serviços.

Desta forma na terceira iteração da arquitectura proposta foram incluídas um conjunto de funcionalidades no módulo cliente que ajudam a minorar a dependência de um único fornecedor de serviços *cloud*. Para isso, foram desenvolvidos clientes para serviços de armazenamento *cloud*. O que nos permite fornecer ao utilizador opção sobre o serviço de armazenamento a utilizar e é particularmente interessante para estudo dos serviços de armazenamento existentes nas plataformas *cloud*. De seguida serão apresentados os serviços SaaS para armazenamento *cloud*, que serviram de base para este desenvolvimento:

Box.net

Box.net [51] é um dos muitos serviços existentes que permite o armazenamento de dados sobre *cloud*, bem como o acesso e partilha destes. De forma a facilitar esta partilha foram desenvolvidas diversas aplicações para as mais variadas plataformas, como por exemplo, Android, Iphone, Ipad. Disponibiliza ainda diversos tipos de *Web Services*.

Nesta plataforma os dados são guardados em diferentes servidores espalhados geograficamente, o que permite a protecção de dados contra diversos tipos falhas. Todas as transmissões entre o cliente e o serviço de armazenamento são protegidas usando Secure Sockets Layer (SSL), de forma a garantir a segurança destas contra terceiros.

Dropbox

Dropbox [52] Trata-se de outro serviço de armazenamento e publicação de dados. Para armazenar os dados usa o serviço S3 da Amazon visto este estar implementado sobre uma infra-estrutura desenhada para suportar tratamento de falhas e protecção dos dados dos clientes. Possui aplicações cliente para diversas plataformas, através destas permite gestão de ficheiros e possui mecanismos de notificação de mudanças dos dados armazenados. Fornece também uma interface para *browsers* Web. De forma a garantir a privacidade dos dados, nas comunicações entre o cliente e o servidor, é usado o protocolo de SSL. Para armazenar os dados em disco é usado o algoritmo de cifragem *Advanced Encryption Standard* (AES).

Estes serviços foram escolhidos pois disponibilizam uma interface REST que podem ser usadas pelos módulos clientes para armazenar os dados das mensagens a enviar. Estes serviços podem ser usados para realizar as mesmas operações que seriam feitas pela parte de armazenamento presente no módulo de *Relay*. Para isso, foram definidas um conjunto de acções que deveriam ser suportadas pelos clientes destas arquitecturas:

- **Criar pasta:** Uma vez que estes serviços funcionam numa estrutura de ficheiros é de todo conveniente agrupar os ficheiros pertencentes aos módulos clientes dentro de uma mesma pasta.
- **Eliminar pasta:** Esta acção é útil para limpeza dos ficheiros armazenados no serviço de armazenamento, quer durante o processo de *logout*, quer durante o processo de *login*. O módulo cliente ao invés de tentar eliminar ficheiros das mensagens enviadas um a um, limita-se a apagar a pasta do serviço de armazenamento. Uma vez que um cliente pode falhar de forma inesperada, esta é na verdade a primeira operação invocada por um módulo cliente quando este se liga a um determinado serviço de armazenamento.
- **Criar ficheiro:** Através desta acção o cliente pode publicar os dados de determinada mensagem. Como resultado são devolvidos os dados necessários para que o ficheiro seja publicamente lido pelo cliente destino e apagado pelo cliente origem.
- **Obter ficheiro:** Dado o endereço público de determinado ficheiro, este pode ser lido através desta acção.
- **Apagar ficheiro:** Possibilita apagar um ficheiro temporariamente colocado no serviço de armazenamento.

Depois de estas acções terem sido definidas, foram implementados clientes para os serviços de armazenamento acima apresentados. Ao mais alto nível foi implementada uma estrutura que permite enviar as mensagens, de forma aleatória, pelos serviços de armazenamento configurados. Com este mecanismo os ficheiros DICOM e os resultados das pesquisas são repartidos em várias mensagens enviadas através de vários serviços de armazenamento o que pode reduzir limitações de taxa de transferência comummente colocadas por certos serviços de armazenamento.

Como é possível inferir pela estrutura apresentada, existem múltiplos parâmetros relacionados com a configuração do módulo cliente que podem ser configurados, como por exemplo, informações relacionadas com a localização do serviço de *Relay* e parâmetros de configuração das contas dos serviços de armazenamento.

Para maior familiarização com a estrutura do módulo cliente, na Figura 25 encontra-se representada a estrutura de pacotes existente. Como se pode observar, existe uma separação clara da estrutura destinada ao envio e recepção da informação de mensagens propriamente ditas e da estrutura de controlo para notificação de chegada destas.

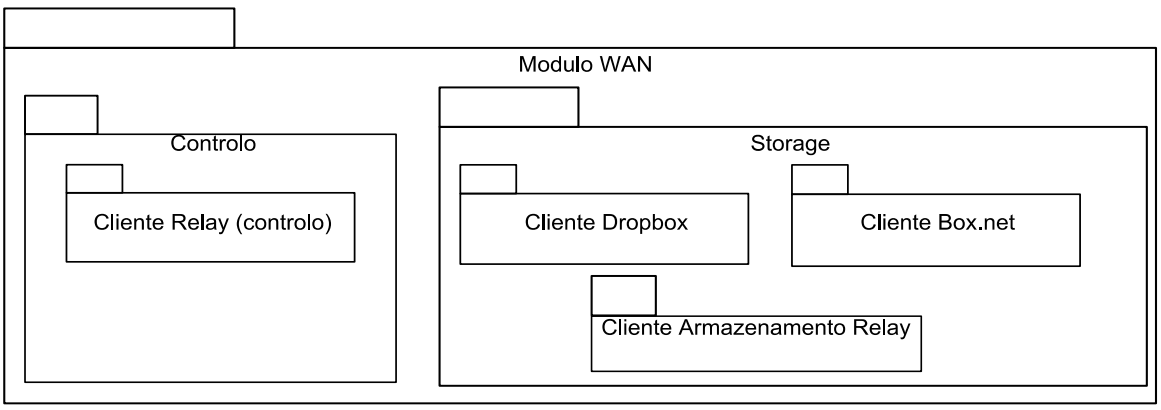

Figura 25. Diagrama de pacotes do módulo Cliente

Uma vez que cada cliente pode ter diferentes serviços de armazenamento configurados, que podem ser distintos dos configurados por outros clientes da mesma comunidade e, para que dois clientes comuniquem, é necessário que os dados sejam acessíveis de forma pública. Dado o carácter confidencial dos dados a enviar, foi necessário implementar mecanismos que garantam a privacidade dos dados.

Conceito de segurança.

Da bibliografia, surgem algumas indicações de que tipo de segurança deve ser implementada num serviço desta natureza, em que as informações a transportar são provenientes de entidades que têm de garantir o sigilo dos dados:

- A partilha de dados deverá ser autorizada pelo responsável da informação.
- Os dados armazenados no serviço devem permanecer confidenciais, isto é o fornecedor de serviços *Cloud* não devem ter a capacidade de comprometer a confidencialidade destes [20].

Até ao momento, a única forma de segurança apresentada no módulo focou-se na autenticação dos clientes com o intuito de descriminar estes e as comunidades ao qual pertencem. De forma a garantir que não são feitos acessos indevidos ao sistema depois de fornecidas as credenciais ao serviço, todos os pedidos devem fornecer o *token* de acesso da sessão do cliente para garantir a protecção deste *token* e dos dados enviados do Cliente para *Relay*, é utilizado HTTPS durante os pedidos.

Outro local onde deverá ser garantida confidencialidade dos dados deverá ser nas notificações do serviço *Relay* para o cliente. Neste ponto, o serviço Channel não fornece qualquer garantia de confidencialidade. Com objectivo de garantir tal confidencialidade é utilizado o algoritmo AES para cifra dos dados. Quando o cliente efectua o *login* a uma comunidade, é fornecida a respectiva chave AES e esta transportada sobre HTTPS.

Como mencionado, uma fraqueza da arquitectura apresentada até ao momento, é o facto dos dados das mensagens se encontrarem acessíveis de forma pública nos serviços de armazenamento *cloud*. Com vista a contornar esta situação, na arquitectura proposta foram descriminadas as chaves de cifragem das mensagens de controlo, das chaves das mensagens de dados. Assim, enquanto as chaves das mensagens de controlo são fornecidas pelo serviço de *Relay*, as chaves para cifragem dos conteúdos das mensagens têm de ser previamente partilhadas pelos clientes que se desejem comunicar. Esta solução permite proteger todas a mensagens contra terceiros e assegura que os dados dos pacientes são mantidos confidenciais do próprio serviço de *Relay*.

Uma questão que pode surgir desta abordagem é: porque o algoritmo AES? Dos serviços de armazenamento estudados este algoritmo é diversas vezes referenciado e uma vez que se trata de um algoritmo de cifra de simétrica permite cifragem de grandes quantidades de dados com um elevado desempenho. Foi assim considerado adequado para garantir a confidencialidade dos dados.

Compressão

Uma operação interessante do ponto de vista do desempenho do sistema e, antes de proceder à cifragem das mensagens, é a compressão de dados. Dependendo dos dados das mensagens a enviar, existem alguns tipos que podem beneficiar da redução do volume de dados com recurso a técnicas de compressão.

Tipicamente, conjuntos de dados formados em texto são bons candidatos para aplicar estas técnicas. Este é o caso de todas as mensagens a serem enviadas pelo sistema apresentado, excepto as do tipo *File Response*. Assim, antes de serem cifradas as mensagens são comprimidas usando o algoritmo zip que permite compressão de dados sem perda de informação.

Na fase de compressão de dados os métodos Java também efectuam o cálculo de um número de controlo. Este número foi aproveitado para verificar a consistência dos dados recebidos pelo receptor da mensagem.

Considerações sobre o fluxo de mensagens

Na Figura 26, encontra-se representado de forma sumária o fluxo de mensagens sobre dois clientes. É possível observar, que o conteúdo das mensagens é tratado em diferentes operações do módulo. De forma sumária o tratamento de uma mensagem envolve as seguintes etapas:

- Do lado do transmissor: A mensagem começa por ser enviada para um sistema *multi-thread* para que as restantes operações sejam tratadas de forma paralela. Os dados da mensagem são então comprimidos e cifrados. São posteriormente transmitidos para um dos serviços de armazenamento que se encontre configurado. De seguida será enviada para o serviço de *Relay* com a informação necessária para que o(s) destino(s) a possa(m) ir buscar ao serviço de armazenamento.
- Do lado do receptor: É recebida a notificação da chegada de uma mensagem, esta notificação é então colocada numa fila. Existe uma *thread* que verifica o estado dessa fila, sempre que é encontrada uma nova notificação e o número de *threads* a receber dados é menor que o definido pelo utilizador é lançada uma nova *thread*. A qual começa decifrar a notificação recebida. Assim, passa a possuir a informação necessária para ler os dados de um dos serviços de armazenamento. Após a leitura, decifra os dados da mensagem, o que lhe permite proceder ao processamento da mensagem.

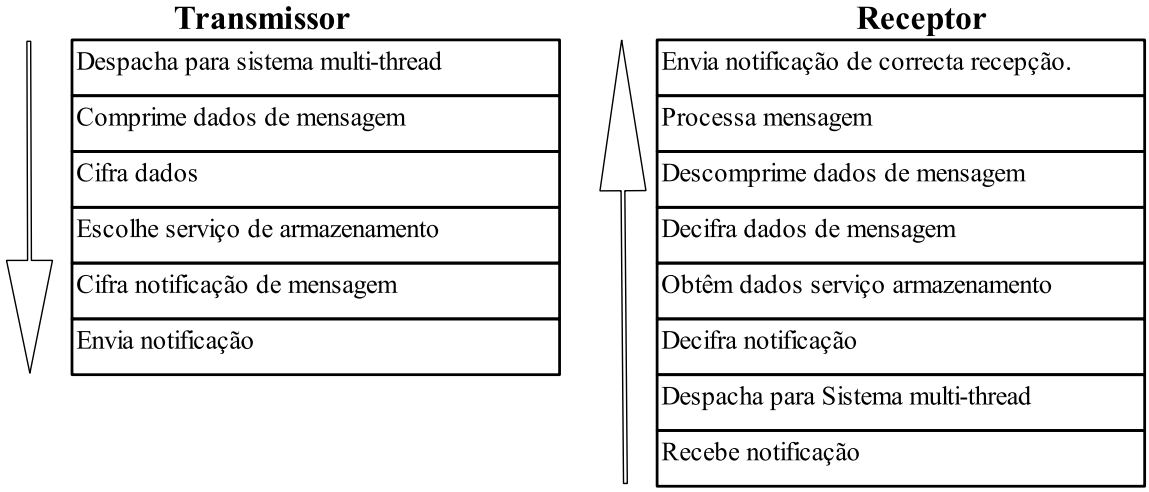

Figura 26. Operações do módulo cliente

Fiabilidade

Como se pode constatar pela Figura 26, na fase final é enviada uma notificação no qual o cliente transmissor é informado sobre a correcta recepção da mensagem.

Uma vez que o processo se encontra divido em múltiplas operações, existem múltiplos locais onde podem ocorrer falhas.

Devido a isso, foram implementadas algumas formas de manter a consistência do sistema. Assim, do lado do *Relay*, sempre que exista um erro a nível do processamento das mensagens é retornado o código do erro através do *status* de um HTTP *response*. Com isto, o módulo cliente será informado sobre o erro ocorrido. Existe também um conjunto de potenciais falhas que podem ocorrer durante o tempo de vida dos clientes, como por exemplo, o envio de uma mensagem para um cliente destino que entretanto falhe deverá ser apagada pelo cliente transmissor.

Por isso, quando um cliente recebe a notificação da falha ou abandono de outro da mesma comunidade, este procede a uma limpeza das mensagens destinada ao cliente destino. Esta limpeza, para além da eliminação de informações locais implica, também pedidos remotos para limpeza de ficheiros nos serviços de armazenamento associados.

3.5 Plataforma móvel

Como enunciado no capítulo anterior, a plataforma Dicoogle mobile encontrava-se totalmente dependente do serviço PacsBroker para efectuar pesquisas. Uma vez que a arquitectura proposta trata de aspectos de comunicação achou-se pertinente a adaptação da plataforma Dicoogle Mobile para suportar comunicações com o serviço de *Relay* desenvolvido. Uma mais-valia considerada foi a introdução de um sistema de pesquisas autónomo nesta plataforma móvel.

Para alcançar tal autonomia foi necessário proceder a uma adaptação das tecnologias existentes na plataforma Dicoogle. Esta adaptação necessitou de endereçar não só questões de *layout* mas também das bibliotecas de suporte à plataforma Dicoogle.

A questão inicialmente colocada foi: Porque é necessária uma adaptação de bibliotecas, uma vez que ambas as plataformas se baseiam em Java? Nas plataformas Android são usadas maquinas virtuais java (JVM) Dalvik optimizadas para os dispositivos móveis [53]. Enquanto, a plataforma Dicoogle foi desenvolvida para máquinas virtuais da Oracle. Assim, devido a questões de dependências das bibliotecas existentes, e para alcançar uma certa autonomia na plataforma *Mobile*, foi necessário proceder a um estudo e conversão das bibliotecas que suportam a plataforma Dicoogle.

Começou-se então pelo estudo da biblioteca dcm4ch responsável por todas as operações associadas às estruturas de dados em formato DICOM. Devido a complexidade da norma, a estrutura do dcm4ch encontra-se dividida em diversos módulos. O módulo central desta biblioteca suporta a extracção de conteúdos presentes nos ficheiros DICOM. Este módulo suporta pois a extracção das imagens existentes em objectos DICOM e para isso é usado um conjunto de bibliotecas nativas da máquina virtual da Oracle. Para resolver essa questão, recorreu-se ao trabalho previamente desenvolvido, nomeadamente um jogo [54] *open-source* que havia sido desenvolvido para JVM Oracle e que mais tarde tinha sido convertido para Android. O autor deste projecto necessitou de desenvolver todo um conjunto de métodos existentes nas JVM Oracle para suportar tal jogo em ambiente Android. Tal aproximação foi efectuada com o núcleo da biblioteca, tornou-se assim possível obter os dados presentes nos ficheiros DICOM em ambiente *mobile*.

Como previamente referido, para indexação e suporte a pesquisas locais sobre dados DICOM, a plataforma Dicoogle usa o indexador Apache Lucene. Existem versões deste que são compatíveis com Android e outras que não. A versão 3.0.3 do Lucene é compatível com Android e por isso foi escolhida para integrar a solução *Mobile*.

Com as duas ferramentas enunciadas e com o trabalho previamente existente na plataforma Dicoogle foi possível pensar em indexar e pesquisar dados imagiológicos em repositórios detidos por dispositivos móveis. Para que estas capacidades pudessem ser usadas por parte dos utilizadores foi necessário adaptar a interface anterior do Dicoogle Mobile.

Depois desta integração concluída, foi então necessário direccionar o estudo para as tecnologias que poderiam permitir à plataforma móvel comunicar com a restante rede Dicoogle. Para isso começou-se por analisar a viabilidade de utilizar a plataforma Jgroups nas comunicações intra-institucionais. Infelizmente, não foi possível integra-lo com a plataforma *Mobile*. De seguida foi estudada a possibilidade de integrar a plataforma móvel com o cliente do serviço de *Relay* para suportar comunicações em WAN. Neste ponto, apenas foram consideradas as notificações do tipo *polling* uma vez que para utilizar *publish-subscribe* existia uma dependência do HtmlUnit e este mostrou-se incompatível com Android. Mais ainda, por uma questão de simplificação também se deixou de parte o suporte de plataformas de armazenamento em *cloud* (Dropbox e Box.net), utilizando exclusivamente o GAE para tal finalidade. Com algumas alterações foi possível ligar o Dicoogle Mobile ao serviço de *Relay* desenvolvido e obter a lista dos utilizadores ligados a este.

Em fase de testes com pesquisas deparamos com problemas na biblioteca usada para converter os pedidos em XML (dom4j [55]) e, para resolver estes, foi necessário usar uma versão previamente adaptada desta biblioteca para a plataforma Android [56].

Em suma, foi necessário adaptar a plataforma móvel para permitir uma certa autonomia mas também um conjunto de bibliotecas utilizadas. Assim, a versão Dicoogle Mobile passou a suportar um conjunto de funcionalidades só disponibilizadas anteriormente na versão desktop.

Além disso, esta plataforma também foi adaptada para permitir comunicações inter-hospitalares recorrendo para isso ao serviço de *Relay* desenvolvido. Tal processo também foi importante para ajustar e validar o serviço de *Relay* em ambientes móveis.

3.6 Notas finais da arquitectura proposta

Como é possível constatar a arquitectura proposta possui um conjunto mecanismos bastante dependestes das tecnologias escolhidas. Desta forma uma abordagem iterativa auxiliou a obter a arquitectura desejada tendo em conta um conjunto de requisitos, evoluções tecnológicas e conhecimento adquirido sobre estas.

Desta forma também foi considerado pertinente uma descrição do trabalho efectuado tendo por base uma descrição ela própria iterativa.

4 Resultados

Neste capítulo serão apresentados os resultados obtidos durante a elaboração do módulo de comunicações criado e também a análise destes. De forma a estudar e melhorar o desenvolvimento do novo módulo, este foi testado e analisado no que respeita aos tempos de pesquisas sobre múltiplos clientes, bem como em relação ao tempo gasto para transmissão de ficheiros DICOM. Estes valores são especialmente importantes uma vez que determinam se o desempenho dos processos implementados é aceitável em cenário de utilização real. Os estudos foram efectuados ao longo das diversas iterações da arquitectura proposta.

Na maioria dos testes efectuados, os clientes encontraram-se em execução em instâncias de máquina virtual alocada nos servidores do *datacenter* do Grupo de BioInformática da Universidade de Aveiro[6]. A máquina disponibilizada possuía 4 GB de memória RAM e um processador a 2.67 GHz. Um parâmetro importante nos testes realizados é a velocidade de acesso à Internet. Infelizmente, na rede do Campus da Universidade a largura de banda disponibilizada aos clientes é bastante variável dependendo bastante da utilização da infra-estrutura por parte de terceiros, pelo que não foi possível especificar este parâmetro com rigor. Uma vez que a largura de banda é um dos parâmetros que mais influencia os valores obtidos este foi estimado recorrendo ao website de speedtest.net chegando-se aos valores de 17,72 Mb/s para download e 44,35 Mb/s para upload.

4.1 Primeira Iteração

Nesta secção serão apresentados os resultados obtidos da primeira iteração do módulo implementado para a arquitectura proposta.

4.1.1 Divisão de tempo pelas fases de processo.

No início de análise que visava proceder à optimização do envio de mensagens sobre o módulo de comunicações WAN para imagem médica, começou-se por dividir o processo de transmissão de mensagens em diversas fases, que foram nomeadas de acordo com o seu objectivo e encontram-se descritas na Tabela 11.

Nome	Descrição
Envio	Tempo necessário para enviar uma mensagem, desde do momento em que esta chega ao modulo de transmissão WAN até que é efectivamente convertida num pedido HTTP.
Transmissão	Tempo despendido na transmissão da mensagem em ambiente WAN, ou seja desde que a mensagem é enviada do cliente até o *Relay*, somado ao tempo necessário para que a mensagem seja enviada do serviço *Relay* até ao cliente destino.
Armazenamento	Tempo ocorrido, desde que os dados de uma mensagem chegam ao serviço de *Relay* até que estes são efectivamente gravados.
Inactivo	Tempo necessário para que uma mensagem gravada no serviço de *Relay* seja pedida a este.
Leitura	Tempo de leitura dos dados da mensagem do serviço de armazenamento.
Processamento	Tempo que uma mensagem que chega ao cliente destino demora a ser tratada por este.

Tabela 11. Fases do processo de envio de mensagens

Na Figura 27 encontram-se representadas estas fases para um processo de pesquisa efectuado sobre 3 clientes WAN, cada um contribuindo com 20662 resultados que são retornados

[6] Á excepção dos testes apresentados na secção 4.5.

em 14 mensagens do tipo *Query Response*. Antes de apresentar e analisar os resultados é necessário efectuar algumas considerações sobre os valores apresentados:

- Uma vez que o módulo cliente procede ao tratamento das mensagens recorrendo a *threads* existe um certo grau de paralelismo. Os valores aqui apresados foram medidos durante o tratamento de tais mensagens. Desta forma, as medições efectuadas apesar de se reflectirem no tempo total das transmissões não o medem.
- Com o objectivo de efectuar as medições de tempos de cada processo, os dados dos tempos foram também eles armazenados e transmitidos o que, por muito insignificativos que estes sejam, acabam por contribuir para as medições efectuadas.

Apesar das nuances referidas os valores medidos inicialmente identificaram claramente as fases do processo que deveriam ser optimizadas. Olhando para o gráfico da Figura 27, verificamos que na versão inicial do serviço de *Relay* grande parte do tempo despendido era devido ao período que as mensagens ficariam à espera de serem pedidas pelo cliente destino. Tal constatação confirma as indicações fornecidas pela documentação da Google, i.e. que um grande conjunto de pesquisas sobre o serviço Datastore é algo bastante ineficiente. Esta observação conduziu ao uso de metadados para redução do número de pesquisas, como previamente mencionado.

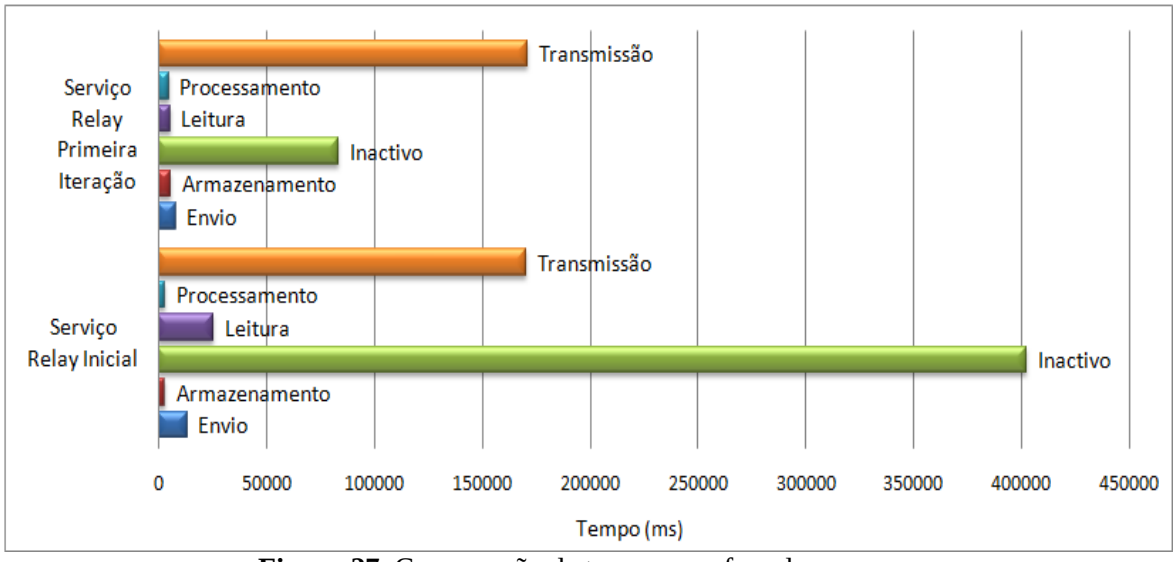

Figura 27. Comparação de tempos por fase do processo

Como é possível constatar pela análise da Figura 27 na primeira iteração da arquitectura proposta o tempo que as mensagens ficavam inactivas no serviço Datastore foi significativamente reduzido à custa de um aumento do tempo necessário para armazenar dados. Pela análise da figura também é possível constatar que os tempos de leitura e envio foram ligeiramente reduzidos a custa de uma política de tratamento de *threads* ligeiramente alterado. Esta alteração viria mais tarde a ser anulada pois em determinadas situações ocorria *deadlock* no módulo cliente. É ainda possível constatar que o tempo de transmissão se mantém praticamente inalterado uma vez que este depende da tecnologia e essa se manteve a mesma.

Na Figura 28 são ainda apresentados os tempos das pesquisas efectuadas, com os resultados provenientes de 3 clientes distribuídos. Foi possível observar em que medida a mudança na estratégia de obtenção dos dados das mensagens, temporariamente armazenados no serviço de *Relay*, contribuíram para a melhoria dos tempos das pesquisas efectuadas. De forma a garantir que ambos os testes se encontravam em iguais circunstâncias ambientais, estes foram efectuados em simultâneo.

Como é possível observar pela Figura 28 o tempo necessário para pesquisas foi significativamente reduzido, em especial nos casos mais críticos das pesquisas com grande número

de resultados. Nestes últimos podemos verificar que o tempo total reduziu praticamente para metade. Também podemos verificar que as pesquisas em WAN são perfeitamente exequíveis pois 61 mil resultados demoram menos de 15 segundos a serem obtidos.

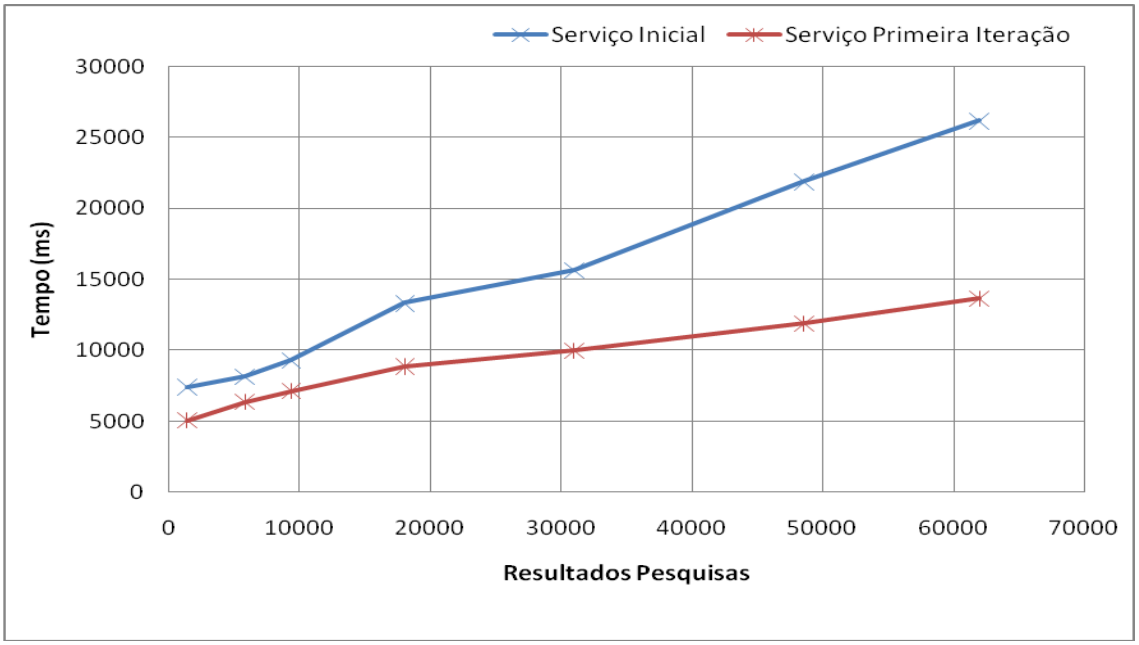

Figura 28. Tempos pesquisas com resultados distribuidos por três clientes

Uma vez que não se sabe *a priori* quantos clientes será necessário suportar, outro parâmetro que interessa estudar é a influência do número de clientes (i.e. peers) existentes na rede no tempo de pesquisa. Os resultados do estudo efectuado para esta relação encontram-se representados na Figura 29.

Figura 29. Relação tempo de pesquisas com número clientes

Chama-se a atenção para o facto do teste com 2 Clientes se encontrar limitado a 30480 resultados. Uma vez, que esse foi o tamanho do repositório local usado para testes.

Como é possível observar pela Figura 29, o aumento do número de clientes diminui o tempo necessário para obter o mesmo número de resultados. Este facto pode ser explicado pelo aumento de paralelismo, visto que os clientes irão processar as pesquisas efectuadas independente dos restantes.

4.1.2 Tempo de transmissão de ficheiros

Analisados os tempos associados ao processo de pesquisa, é necessário efectuar estudo similar para a transmissão de ficheiros de imagem médica. Assim, tendo em vista uma análise comparativa entre os tempos da primeira iteração do módulo desenvolvido e a versão inicial do serviço, são apresentados os tempos necessários para transmissão de ficheiros com diversas dimensões na Figura 30.

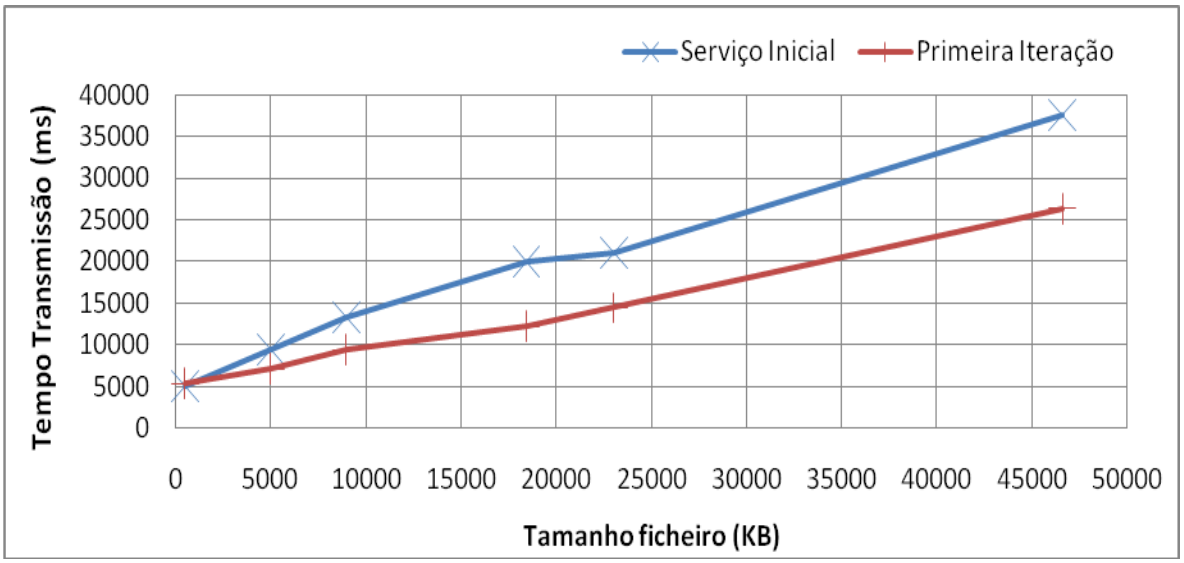

Figura 30. Tempos de transferência ficheiros Primeira iteração *VS* Inicial

Como é possível constatar pelos resultados obtidos o tempo necessário para transmissão de ficheiros na primeira iteração também foi significativamente reduzido quando comparado com a versão inicial. Os resultados observados possuem uma explicação semelhante a descrita na analise do gráfico da Figura 28, uma vez que as mensagens do tipo *Query Response* são tratadas no serviço de *Relay* da mesma forma que as mensagens do tipo *File Response*.

4.1.3 Interface primeira iteração

Na Figura 31 é apresentada a interface do administrador do sistema para gestão de utilizadores. Pode-se observar que foram definidos dois tipos de utilizadores e que os administradores podem geri-los bem como verificar quais se encontram com sessão activa.

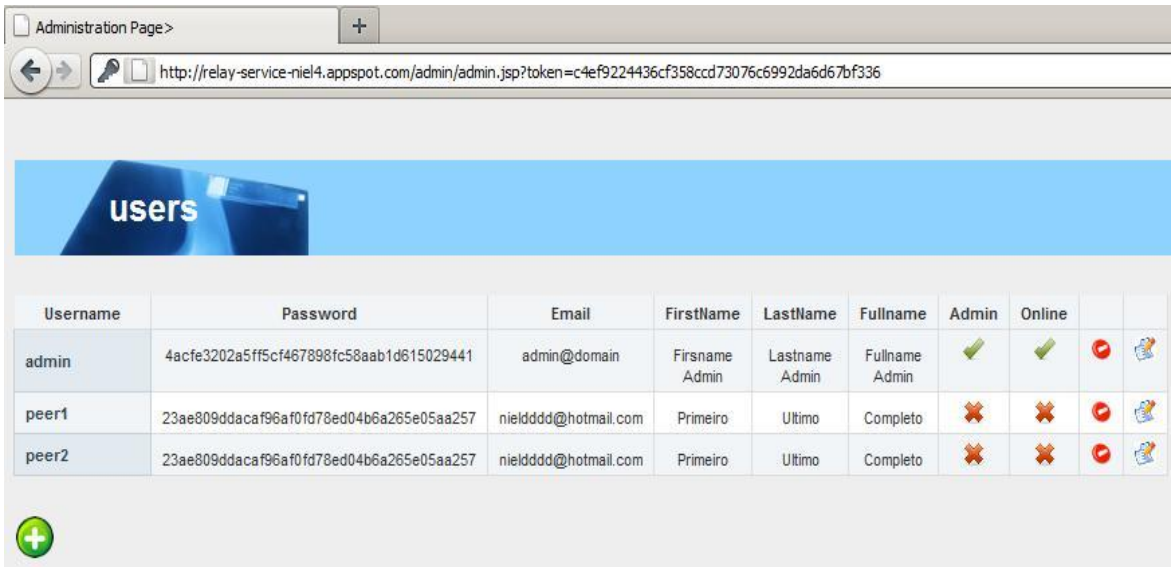

Figura 31. Interface Administração primeira iteração

É possível notar que esta interface permite ao administrador efectuar um conjunto de acções de uma forma muito simplificada. No entanto, esta viria a ser abandonada uma vez que se considerou que seria pouco exequível obrigar o administrador a registar todos os clientes presentes na rede e não é isso que tipicamente acontece com uma rede *peer-to-peer*.

4.2 Segunda Iteração

Nesta subsecção são apresentados os resultados obtidos com a segunda iteração da arquitectura proposta.

4.2.1 Pesquisas sobre múltiplos clientes

Como mencionado, o principal mecanismo modificado durante a segunda iteração foi a forma como os clientes são notificados pelo serviço de *Relay* relativamente às mensagens a si destinas. Assim, na Figura 32 são apresentados os tempos das pesquisas usando o mecanismo de *polling* como forma de notificação.

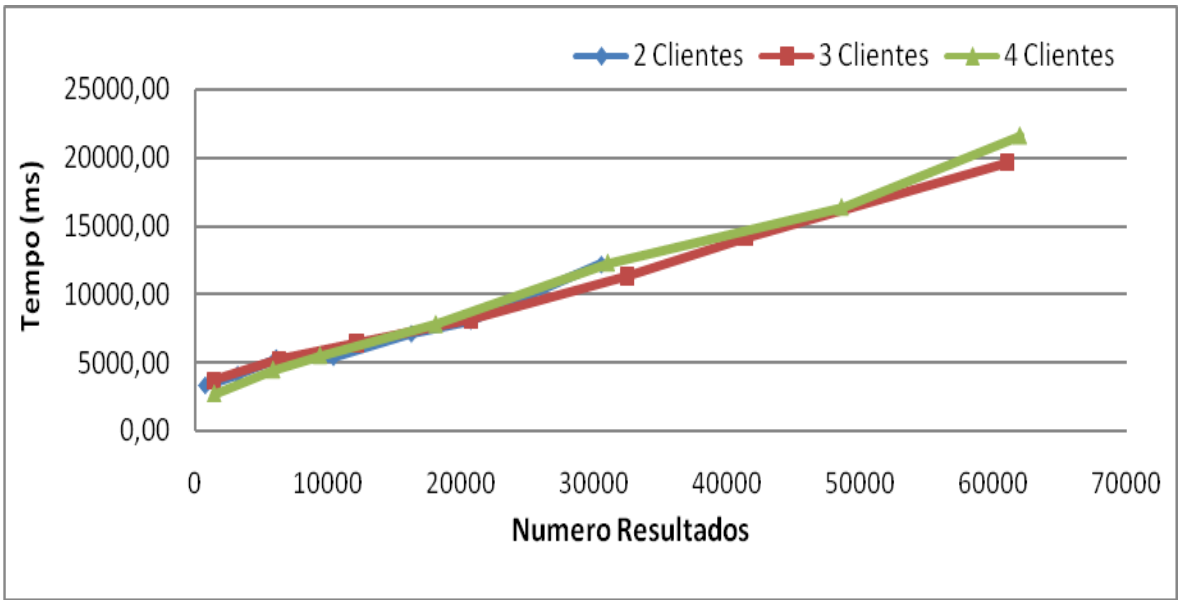

Figura 32. Segunda iteração notificação do tipo *Polling*

É possível observar pela figura pela Figura 32 que na segunda iteração o tempo das pesquisas não foi tão bom como os apresentados para a primeira iteração (Figura 29), isto pode ser explicado quer por se ter reposto o mecanismo de gestão de threads inicial, quer pela divisão da transmissão em parte de dados e parte de notificação. É ainda possível observar que se perdeu paralelismo na transmissão de informação uma vez que os tempos se mantêm praticamente inalterados com a variação de número de clientes.

Os valores obtidos com o serviço Channel, o outro mecanismo considerado para notificações, encontram-se representados na Figura 33.

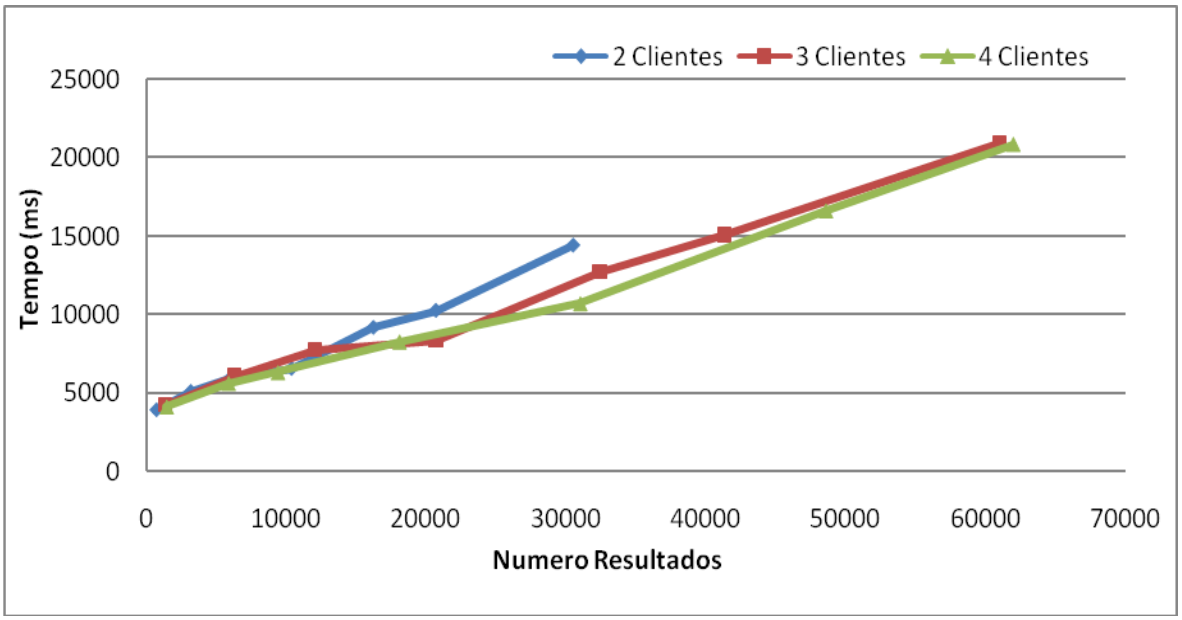

Figura 33. Segunda iteração notificação do tipo *Publish-Subscribe*

Pelos resultados obtidos na Figura 32 e na Figura 33 indicam que para o nosso sistema os mecanismos de notificação estudados apresentam um desempenho muito semelhante com o aumento do número de clientes. Na Figura 34 encontram-se representadas as diferenças entre os valores obtidos para os diferentes mecanismos de *Publish-subscribe* e *Polling*.

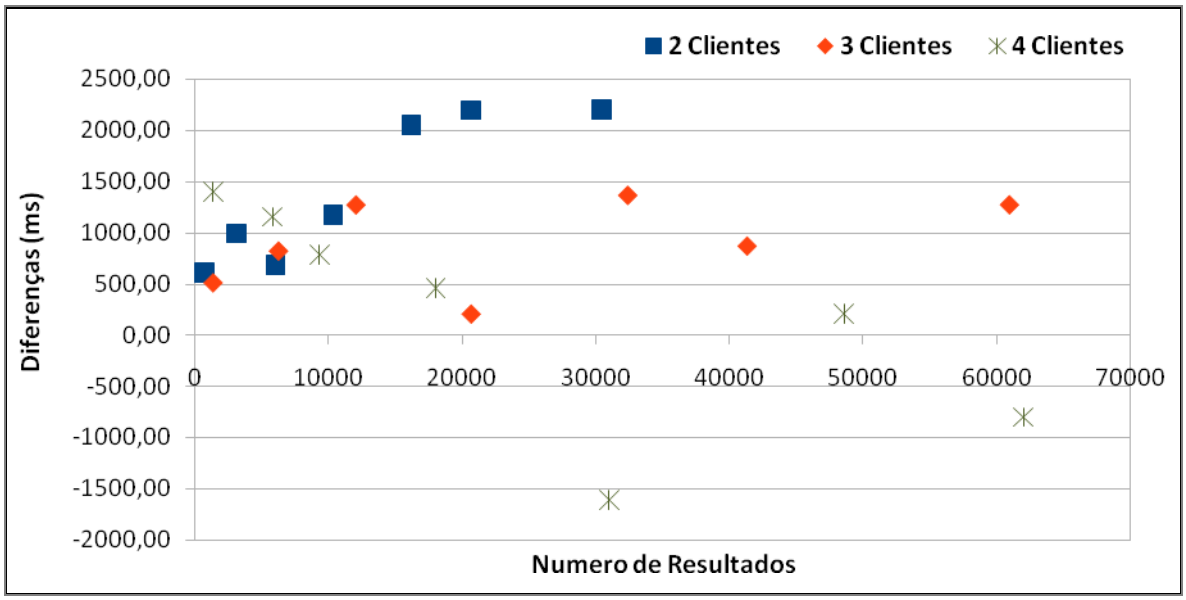

Figura 34. Diferenças entre pesquisas com Channel e com Polling

Pelo gráfico das diferenças apresentado na Figura 34 é possível observar que as diferenças de desempenho entre os dois mecanismos de notificação considerados são bastante reduzidas. E parecem ser diminuir com o aumento do número de clientes.

4.2.2 Tempo de envio de ficheiros

De forma a completar o estudo dos mecanismos de notificação, na Figura 35 são apresentados os valores para as transferências de ficheiros, entre dois clientes. Estes resultados estão de acordo com o observado na Figura 34 onde, para transmissões entre dois clientes, na notificação por *Polling* se obtêm valores muito próximos a notificação com *Publish-Subscribe* do serviço Channel.

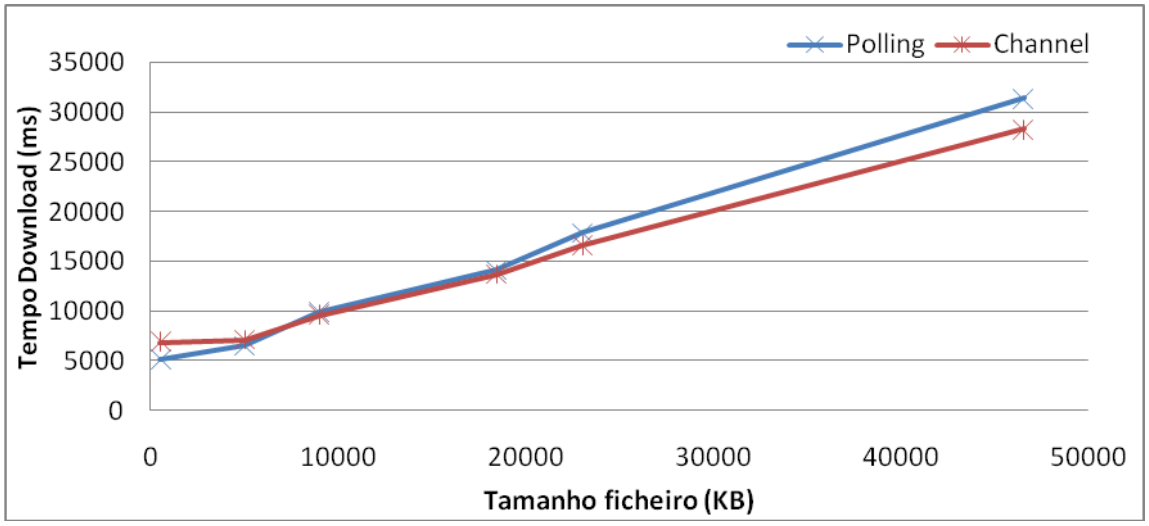

Figura 35. Transferência de ficheiros, segunda iteração

Como previamente mencionado, o serviço de *Relay* encontra-se alocado numa infra-estrutura que lhe permite crescer consoante os pedidos que lhe são endereçados. Para que esse crescimento ocorra o serviço é alocado em diversas instâncias, parâmetro de especial interesse uma vez que constituirá um factor de pagamento do serviço utilizado.

No gráfico da Figura 36 é apresenta a evolução do número de instâncias depois dos testes da Figura 35 terem sido efectuados. É de notar que o número de instâncias apresentado é fraccionário uma vez que é resultado da média de uma repetição de testes.

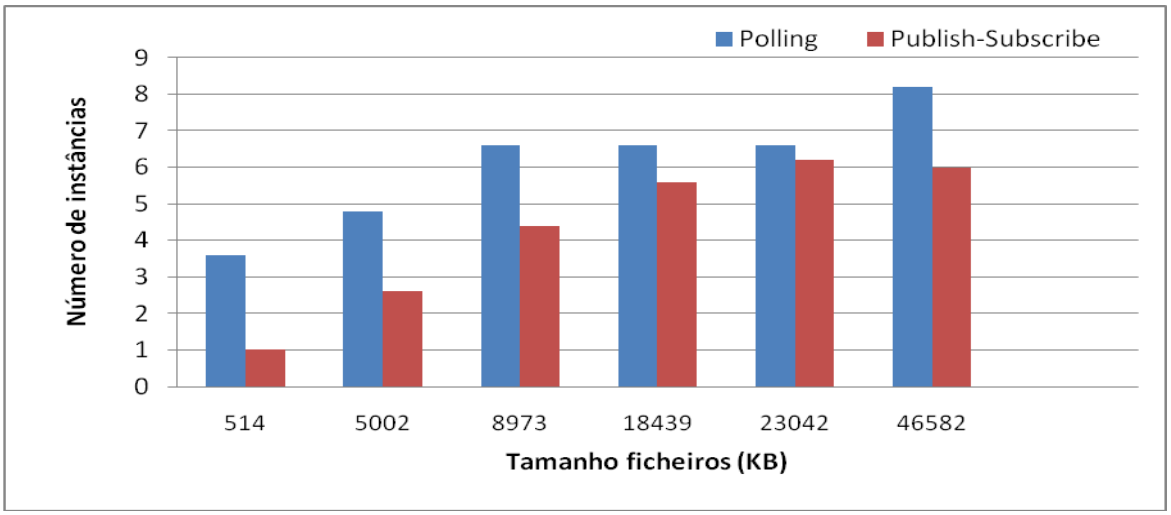

Figura 36. Relação entre número de instâncias e métodos de notificação

Como é possível observar pela Figura 36 a quantidade de instancias que é necessário utilizar com o mecanismo de *publish-subcribe* é reduzido quando comparado com o método de notificação baseado em *polling*. Outra constatação pertinente é que o número de instâncias necessárias para o serviço de *Relay* não cresce uniformemente em relação a tamanho do volume de dados gerado pelos pedidos efectuados.

Assim, apesar de se ter verificado uma ligeira perda de desempenho em relação aos valores da primeira iteração foi possível verificar que com *publish-subscribe* se ganha em termos poupança de recursos. No entanto, também sabemos que o uso de *Publish-Subscribe*, só por si, não é uma alternativa visto que é necessário renegociar o Token de acesso ao canal com o serviço Channel pelo que, na fase de transição, é usado Polling para garantir que o cliente continua a receber as notificações do *Relay*.

Concluindo, esta segunda iteração permitiu-nos constatar que não há um mecanismo de notificação que se destaque do outro em termos de desempenho. Assim, decidiu-se manter ambos na versão final.

4.3 Terceira Iteração

Nesta análise pretendeu-se estudar qual a relação entre o tempo de transmissão dos ficheiros com o serviço de armazenamento usado para o fazer. Na Figura 37 são apresentados os tempos obtidos para transmissões de ficheiros entre dois clientes.

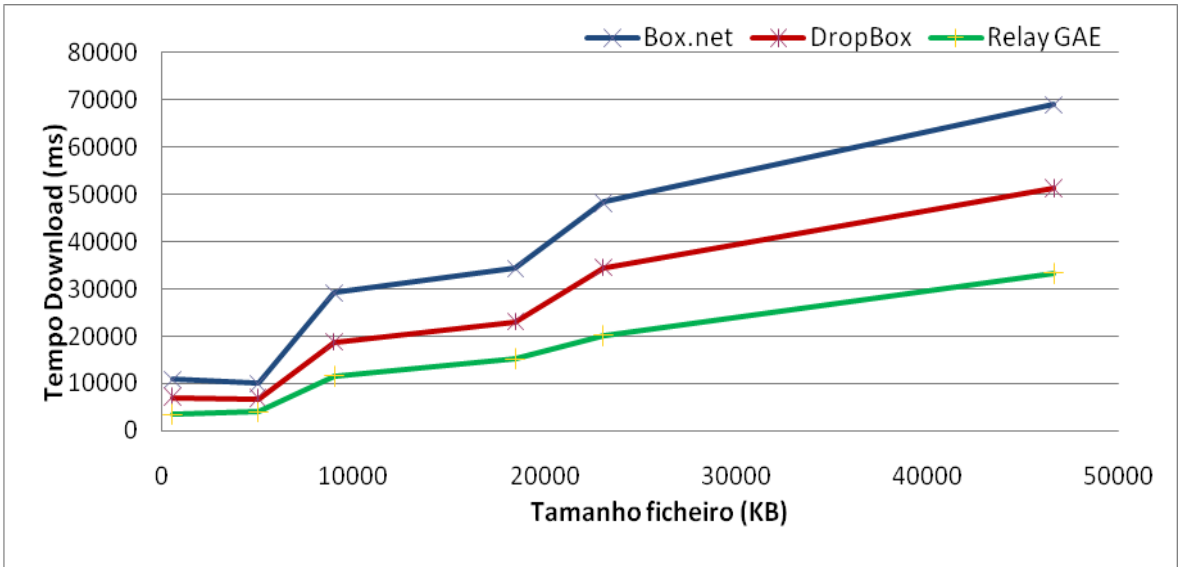

Figura 37. Comparação entre diferentes serviços de armazenamento

A primeira constatação que se pode efectuar a partir da observação do gráfico da Figura 37 é que, para o módulo desenvolvido e para o conjunto de ficheiros estudado, o serviço de armazenamento sobre o serviço de *Relay* baseado em GAE consegue melhores resultados, face aos restantes serviços considerados. Outra constatação pertinente neste gráfico surge quando se observa que o tempo de transmissão de ficheiros com o serviço *Relay* GAE não melhorou relativamente ao apresentado para a segunda iteração (Figura 35) e se encontra com um desempenho muito semelhante ao serviço de *Relay* inicial (Figura 30), mesmo sem este recorrer a qualquer tipo de compressão.

A questão que se coloca é: Porque tal ocorre? Para fornecer uma explicação é necessário recorrer a bibliografia onde é referido que o serviço GAE negocie-a com os Web *browsers* a compressão dos dados dos pedidos [27]. O indica que o factor compressão havia estado presente nas comunicações com o serviço de *Relay* desde da versão inicial.

A segunda questão é: Então porque desenvolver nesta iteração, um conjunto de métodos que efectuam compressão explícita dos dados? Nesta iteração os dados antes de serem transmitidos

são cifrados. Se a compressão fosse realizada depois desta cifra, não seria possível aproveitar a correlação existente nos dados originais o que levaria a uma acentuada perda de desempenho. Outra vantagem dos dados serem comprimidos, de forma explícita, é o facto do dados ocuparem menos espaço quando armazenados temporariamente, o que permite economizar recursos. Além do mais, apesar de ser indicado que o serviço GAE efectua a compressão de dados, nada é referido em relação aos restantes serviços de armazenamento de dados. Pelo que seria injusto comparar serviços que efectuam compressão *a priori* com serviços que não o fazem.

Outra questão considerada pertinente, foi o que ocorreria caso uma imagem fosse transmitida utilizando diferentes serviços de armazenamento. Como foi referido no capítulo anterior, o módulo cliente foi alterado para permitir o envio de fragmentos dos ficheiros pelos diferentes serviços de armazenamento, e de forma aleatória. Na Figura 38 são apresentados os resultados para transmissões de ficheiros sobre diferentes conjuntos de serviços de armazenamento.

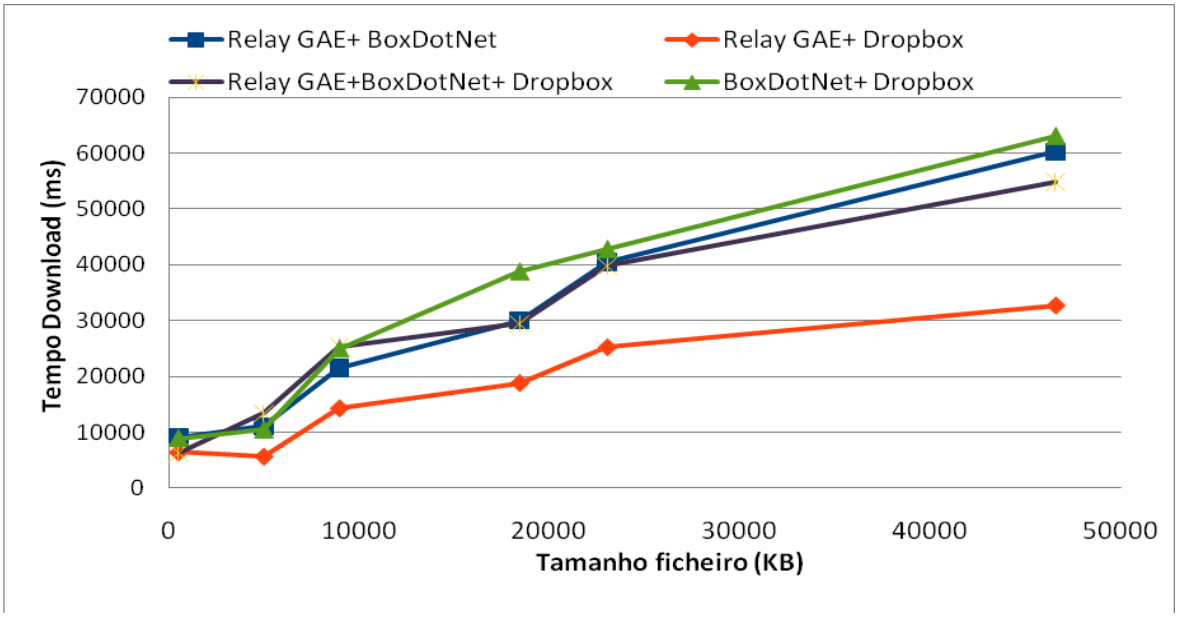

Figura 38. Combinação transferências de ficheiros

Antes de se proceder a estes testes, diferentes cenários haviam sido considerados:

- O tempo obtido seria o pior dos serviços de armazenamento considerado uma vez que as mensagens mais lentas poderiam atrasar todo o processo.
- O resultado final seria obtido por uma média dos serviços de armazenamento considerados.
- Poderia ser alcançado um desempenho que seria uma soma de ambos os serviços uma vez que a divisão do tráfego gerado por diferentes serviços de armazenamento poderia evitar algum tipo de controlo de tráfego existente por parte do fornecedor de serviços de armazenamento em *cloud*.

Como é possível constatar, quando se compara o gráfico da Figura 38 com o da Figura 37, os valores obtidos parecem ser uma média dos valores dos diferentes serviços de armazenamento. Tal facto indica que nem existiu uma melhoria devido a divisão de tráfego nem se piorou devido a um determinado serviço de armazenamento.

De forma a completar o estudo do desempenho obtido com o módulo desenvolvido são apresentados na Figura 39 os resultados das pesquisas efectuadas sobre 3 clientes. Neste gráfico apenas foi considerado o uso do serviço de *Relay* para armazenamento temporário de dados, uma vez que se pretendia comparar o desempenho obtido na segunda e terceira iterações.

Como é possível constatar pela Figura 39 os tempos das pesquisas da segunda para a terceira iteração permaneceram praticamente inalteradas. O que indica que nem a cifra de dados nem a compressão destes contribui-o de forma significativa para uma mudança do desempenho.

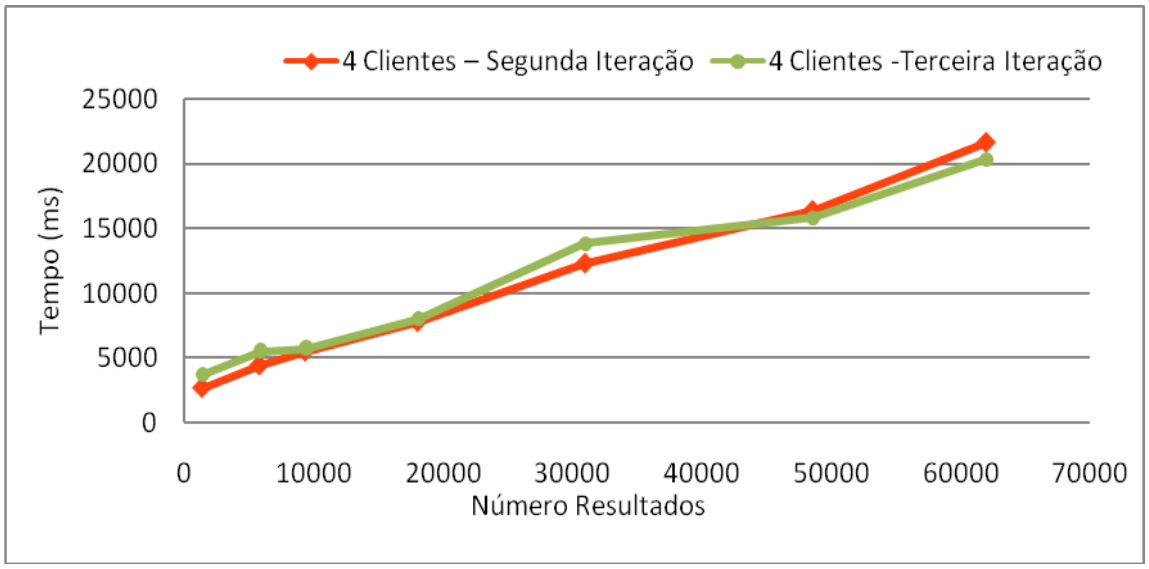

Figura 39. Comparação entre pesquisas segunda e terceira iterações

Depois das indicações bibliográficas e dos resultados obtidos a questão que permanece é: Uma vez que este não se trata de um Web *browser*, será que o módulo cliente efectuou realmente pedidos com compressão? Na Figura 40 é apresentado uma captura do registo dos pedidos efectuados ao serviço de *Relay*. Como é possível constatar, a biblioteca usada para efectuar os pedidos HTTP efectuou a compressão dos dados usando para isso o algoritmo de compressão gzip.

Figura 40. Captura de logs do serviço *Relay*

4.4 Integração

Uma vez que o desenvolvimento do módulo foi efectuado de forma separada da plataforma Dicoogle. Foi feita uma adaptação da arquitectura para permitir a integração do módulo WAN com a plataforma Dicoogle. Tal integração foi feita com sucesso e na Figura 41 é apresentada uma captura da interface gráfica da plataforma Dicoogle depois de ter sido efectuada uma pesquisa com resultados sobre o repositório Local, LAN e WAN.

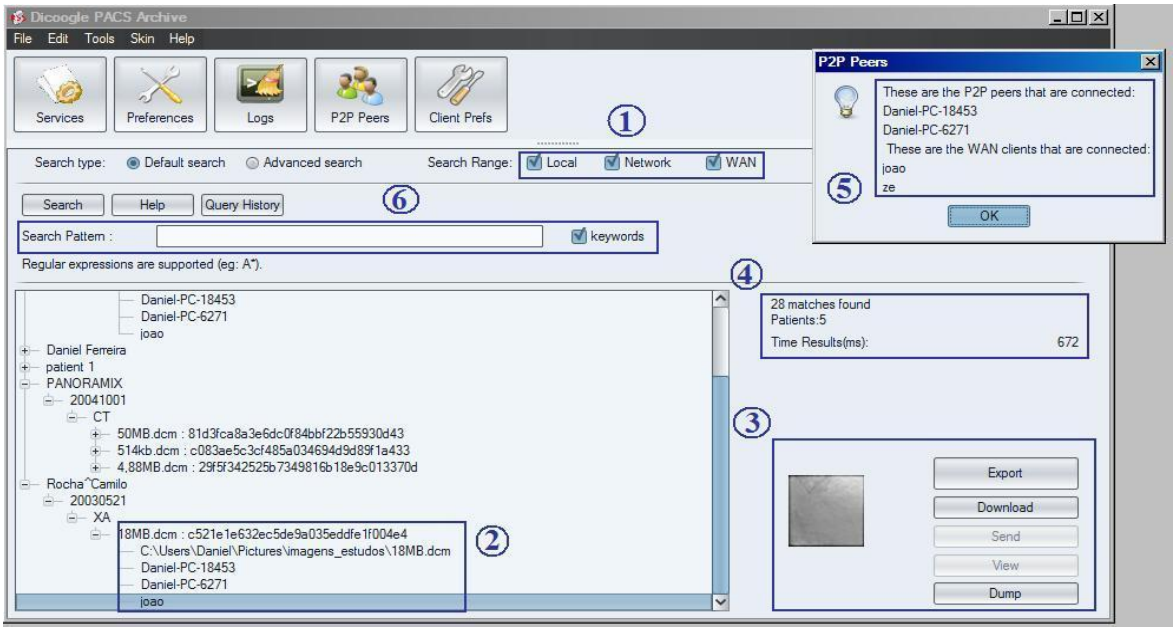

Figura 41. Interface Dicoogle WAN

Na interface da plataforma Dicoogle, apresentada na Figura 41, é possível destacar algumas zonas de especial interesse para o módulo WAN que se encontram numeradas na figura e descritas na Tabela 12.

Número	Descrição
1	Permite seccionar os locais onde se pretende efectuar a pesquisa.
2	Nesta zona são preenchidos os resultados da pesquisa, a zona seleccionada especificamente permite descriminar qual a localização de onde veio o resultado.
3	Local onde é mostrada uma pré-visualização do resultado seleccionado e onde é possível solicitar a transferência do ficheiro.
4	Onde é mostrada estatística sobre a pesquisa efectuada: Quantos resultados foram devolvidos; Quantos pacientes e qual o tempo que demorou a pesquisa.
5	Mostra a lista dos utilizadores que se encontram activos em LAN (via Jgroups), bem como os da comunidade do serviço de *Relay* (i.e.WAN).
6	Expressão de pesquisa propriamente dita.

Tabela 12. Descrição interface Dicoogle WAN

Foi possível efectuar vários testes de utilização da plataforma Dicoogle contendo o módulo WAN. Inúmeros testes foram feitos com sucesso numa rede WAN contendo um número variável de *peers*. Mais ainda, foi possível verificar que o módulo WAN não interfere com a componente de pesquisa LAN e local. Assim, é possível constatar que a arquitectura desenvolvida se adapta perfeitamente às necessidades da plataforma Dicoogle.

Instalador

Como previamente mencionado, uma das razões que levaram a escolha do serviço GAE como plataforma de implementação do serviço de *Relay* deveu-se ao facto deste permitir um serviço simplificado e possuir uma oferta inicial gratuita.

De forma a permitir uma utilização simplificada do serviço App Engine por parte dos utilizadores Dicoogle, foi criada uma aplicação que auxilia nas etapas necessárias para obter uma conta GAE e instalação do serviço de *Relay*, permitindo configurar parâmetros referentes às comunidades do serviço de *Relay* e à publicação deste.

Na Figura 42 encontra-se é apresentada uma captura desta aplicação.

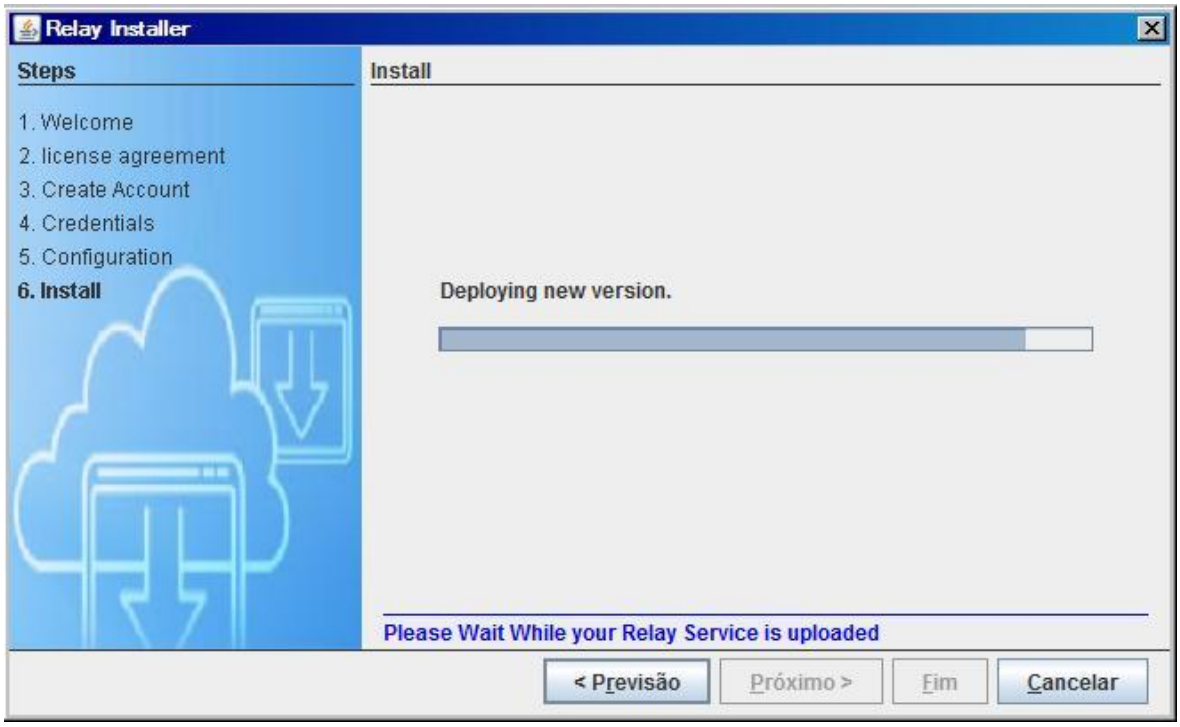

Figura 42. Captura do instalador do serviço de *Relay*

O processo de instalação do serviço de Relay encontra-se dividido em seis etapas descritas na Tabela 13.

Número	Descrição
1	É apresentado um pequeno resumo da funcionalidade do assistente.
2	O utilizador é informado dos termos de utilização do assistente.
3	É descrito como o utilizador poderá obter uma conta GAE.
4	Nesta fase o utilizador informa qual o seu Nome de utilizador, Palavra-chave do Google e qual a localização onde deseja que o serviço de *Relay* seja Implantado.
5	Permite configurar propriedades referentes as comunidades dos serviço de *Relay*.
6	Nesta fase o utilizador é informado de qual o estado da implantação do serviço de *Relay*. Depois desta fase o serviço de *Relay* estará pronto para servir pedidos dos seus clientes Dicoogle.

Tabela 13. Fases instalação serviço de *Relay*

4.5 Plataforma móvel

De forma a testar a arquitectura na plataforma móvel desenvolvida foram projectados e efectuados um conjunto de testes sobre um telemóvel Sony Ericsson XPERIA X8 com Android 2.2 instalado. Este dispositivo possuí 128 MB de memória RAM e um processador a 600MHz. A largura de banda disponível para este dispositivo durante os testes efectuados foi estimada recorrendo ao website de speedtest.net, tendo-se chegando a valores de 4,96 Mb/s para download e 0,35 Mb/s para upload.

Pesquisas locais

O primeiro teste efectuado sobre a plataforma móvel permitiu-nos estudar a relação que existe entre o número de resultados de uma pesquisa local e o tempo necessário para concluir esta.

À semelhança dos testes anteriormente apresentados foi usado um índice contendo informação relativa a 30480 objectos DICOM. Os resultados obtidos encontram-se representados na Figura 43.

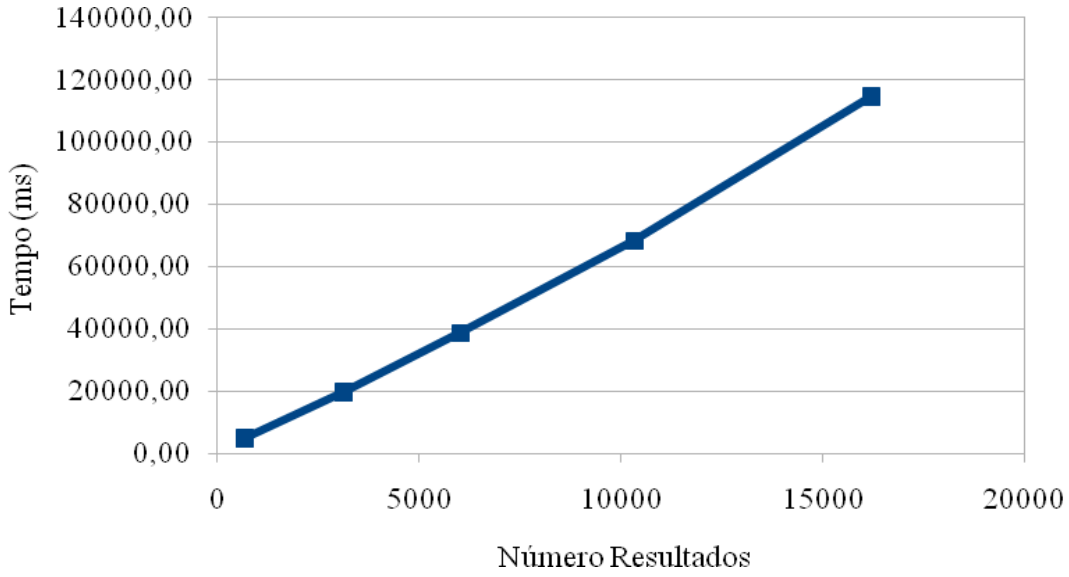

Figura 43. Pesquisas móveis locais

Como se pode observar pela Figura 43, o tempo necessário para efectuar pesquisas sobre a plataforma móvel é bastante aceitável e evolui de forma linear em relação ao número de resultados devolvidos e as limitações computacionais do dispositivo utilizado.

Transferências de ficheiros remotos

De forma a estudar o tempo necessário para efectuar transferências de imagens para a plataforma móvel, também se efectuou testes onde foram solicitados ficheiros de uma plataforma Dicoogle tradicional, que havia sido considerada nas secções anteriores. Os tempos necessários para a transferência de tais ficheiros encontram-se representados na Figura 44.

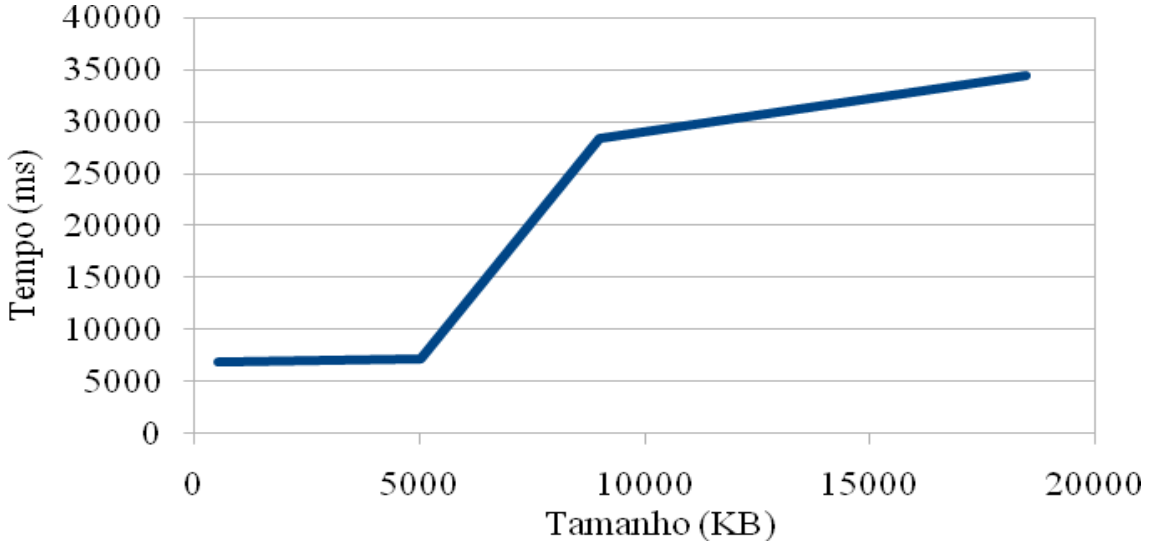

Figura 44. Transferência de ficheiros Dicoogle tradicional para Dicoogle Mobile

Não foi possível efectuar um teste tão completo como nas secções anteriores, o que sugere que ainda será necessário efectuar todo um conjunto de mudanças para adaptar a transferência de ficheiros e pesquisas de dados ao cenário móvel. No entanto, os resultados obtidos permitem perspectivar uma utilização efectiva de tal plataforma móvel num cenário real.

Interface

De forma a permitir pesquisas tanto locais como remotas, assim como transferência de ficheiros e visualização do conteúdos destes, foi necessário adaptar a interface do Dicoogle Mobile antigo e adicionar algumas acções presentes no Dicoogle tradicional.

Assim, na Figura 45 é apresentado o menu principal do Dicoogle Mobile adaptado e na Figura 46 a lista que possibilita visualizar o peers que se encontram activos.

Figura 45. Menu principal Dicoogle Mobile

Figura 46. Lista de peers Dicoogle Mobile

Como se pode visualizar pela Figura 45 existe um conjunto de acções que o utilizador pode efectuar através do Menu principal do Dicoogle Mobile:

- Deslocar-se para o menu de configuração no qual pode configurar a localização do serviço de *Relay* e a localização do repositório local.
- Recriar o índice usado para suporte as pesquisas locais.
- Deslocar-se para um menu onde é apresentado uma lista de peers activos na rede Dicoogle.
- Seleccionar onde pretende efectuar a próxima pesquisa e também ordenar a execução desta.

Depois de efectuada uma dada pesquisa, o resultado gráfico de tal operação encontra-se representada na Figura 47 e na Figura 48. Como é possível observar, para além de serem apresentados dados dos resultados propriamente ditos, também é mostrado ao utilizador uma estatística da pesquisa efectuada e, no caso apresentado, foram encontrados 360 resultados em 16525 milissegundos.

Figura 47. Primeiro nível de selecção

Figura 48. Segundo nível de selecção

Esta interface foi desenvolvida para representar o modelo hierárquico do Dicoogle que tinha sido devolvido para selecção de ficheiros DICOM mas agora para a arquitectura móvel. Assim, em vez de utilizarmos uma árvore, como aquela representada na Figura 41, é usada uma lista que vai alterando o seu conteúdo conforme as selecções efectuadas pelo utilizador.

Quando o utilizador chega ao último nível da selecção é possível descriminar exactamente qual o ficheiro que este pretende visualizar e é mostrado um sumário com as informações do ficheiro em questão. Caso este necessite de ser visualizado remotamente é possível seleccionar a transferência para o dispositivo local. Uma captura da interface onde é mostrado este sumário pode ser vista na Figura 49.

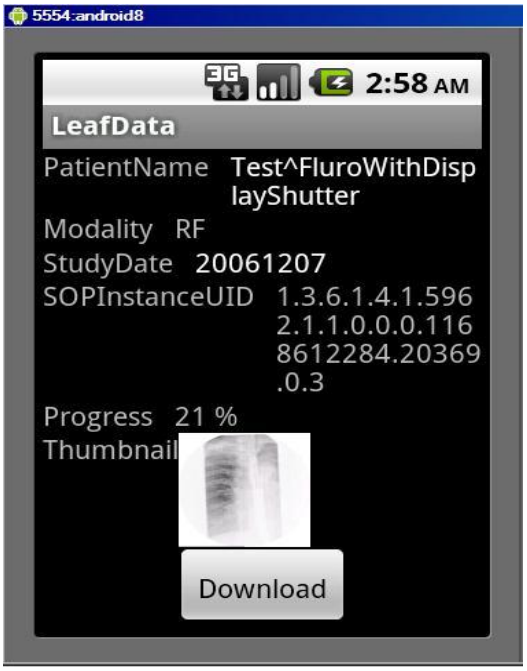

Figura 49. Transferência de um ficheiro para a aplicação móvel

5 Conclusões

Este trabalho surge da necessidade de efectuar pesquisas e acesso remoto a repositórios de imagem médica constituídos por ficheiros DICOM. Pretendia-se que tal fosse feito de uma forma flexível e eficiente. Acreditamos que, no futuro, o tratamento de questões relacionadas com o envio de imagem médica sobre redes de acesso global dependerá da evolução da norma DICOM. No entanto, só por si, a normalização não chega, pelo que é necessário implementar sistemas eficientes que tratem das comunicações a um nível global.

Neste contexto, o paradigma *cloud computing* poderá dar um importante contributo para a partilha de imagem médica uma vez que as grandes infra-estruturas disponibilizadas pelos fornecedores de serviços *cloud* oferecem capacidade de resposta aos requisitos específicos do cenário da imagem médica, nomeadamente a necessidade de manipulação de grandes volumes de dados que necessitam de estar disponíveis 24 horas por dia.

Por parte destes fornecedores de serviços *cloud*, seria interessante um esforço de normalização, tendo-se verificado que já existe algum esforço nesse sentido. Por exemplo, as tecnologias IaaS *open-source* consideradas neste trabalho adoptaram as interfaces da Amazon. Assim, é espectável que, no futuro, exista um maior número de *providers* com interfaces normalizadas de acesso aos serviços disponibilizados. Para além disso, também é espectável que os clientes possam escolher os seus fornecedores de serviço mais focados naquilo que cada serviço *cloud* pode oferecer ao seu negócio, sem ficarem reféns de escolhas tecnológicas.

Assim, apesar das tecnologias que suportam serviços *cloud* se poderem considerar maduras, é questionável até que ponto a forma como os serviços são fornecidos se encontram realmente estáveis. Há algumas questões ainda em aberto como, por exemplo, qual será o posicionamento escolhido pelos fornecedores de serviços *cloud* e mesmo quem serão os fornecedores que continuarão a competir neste paradigma

A mudança para um determinado serviço de *cloud* deve sempre comportar uma análise cuidada dos benefícios, desvantagens e da legislação que regulamenta o negócio de um potencial cliente. Uma vez que é necessário arrendar os recursos consumidos, é vital ainda efectuar uma análise cuidada da quantidade de recursos necessários para que não ocorram surpresas inesperadas aquando do pagamento do aluguer da infra-estrutura utilizada.

É importante referir que *cloud computing* pode representar uma solução para as necessidades identificadas, mas existem mais soluções. No caso especifico do módulo desenvolvido, a escolha tecnológica mais natural seria usar as extensões disponibilizadas pelo Jgroups para comunicações entre diferentes grupos. O serviço de *Relay* baseado em tecnologias *cloud* foca-se assim em aspectos mais de usabilidade e conveniência de utilização para o utilizador final. Desta forma, as escolhas tecnológicas efectuadas dentro do paradigma *cloud* focaram-se naquilo que seria interessante fornecer ao utilizador final em termos de usabilidade. Assim, optou-se por utilizar uma plataforma *cloud* PaaS que nos forneceu uma conveniente base tecnológica para os serviços desenvolvidos. Pelo acompanhamento das tecnologias consideradas foi possível observar que os serviços PaaS se têm dirigido mais para as tecnologias Web e o seu desenvolvimento se tem focado na evolução dos tradicionais servidores aplicacionais, como é o caso do GAE e do serviço recentemente lançado pela empresa VMware denominado Cloud Foundry [57].

Com esta base tecnológica disponibilizada pela plataforma *cloud*, foi possível desenvolver um serviço de *Relay* que pode ser facilmente configurado pelo utilizador final e possui uma arquitectura que consegue ir para além das limitações que seriam impostas por um serviço de *Relay* baseado numa única máquina, aproveitando a flexibilidade e resistência a falhas que é possível obter com uma plataforma distribuída.

A nossa arquitectura também demonstra ser suficientemente flexível para permitir que o utilizador final escolha o serviço de armazenamento que melhor satisfaz os seus requisitos, pois trata-se do componente mais crítico em termos de desempenho no acesso aos dados e, consequentemente, nos custos financeiros envolvidos na sua utilização.

Os *providers* escolhidos para fornecer tal serviço de armazenamento têm arquitecturas semelhantes e as ofertas inicialmente gratuitas permitiram que a nossa arquitectura pudesse ser desenvolvida sem custos de operação.

Á inexistência de custo para é um importante factor de toda a infra-estrutura Dicoogle, especialmente quando se pensa numa escala planetária onde os países subdesenvolvidos não têm recursos para dotar as suas instituições com infra-estruturas PACS e apenas recentemente têm considerado a utilização de sistemas informáticos para assistência ao diagnóstico médico.

Para as instituições destes países, bem como para qualquer outra, todo o trabalho desenvolvido no âmbito do projecto Dicoogle pode representar uma significativa redução de custos com tecnologia. Assim, é importante desenvolver uma análise crítica dos aspectos considerados pertinentes do módulo desenvolvido e fornecer indicações futuras.

5.1 Análise crítica

De forma a providenciar a melhor experiência possível para o utilizador final foram estudadas diferentes formas de realizar determinadas operações. Neste contexto, o módulo desenvolvido demonstrou que podem existir variações significativas de desempenho dependendo da forma como as acções são efectuadas sobre a plataforma tecnológica. Tal verificou-se nos testes efectuados com o serviço Datastore do GAE quando os resultados obtidos na primeira iteração demonstram significativas melhorias face aos do serviço de *Relay* inicial.

Outra conclusão importante retirada dos testes efectuados com o módulo desenvolvido é que diferentes métodos de notificação, que apresentam uma abordagem bastante diferenciada, podem na verdade fornecer o mesmo grau de satisfação para o utilizador final.

A questão que se colocou a quando da realização destes testes foi se estes reflectem realmente o que é espectável em situações reais, tendo por base o conhecimento adquirido dos sistemas actuais de diagnostico. Este tipo de sistemas gerem enormes quantidades de informação mas é questionável a pertinência de os utilizadores efectuarem pesquisas que devolvem centenas de milhares de resultados e até que ponto devem ser realista considerar um grande número de clientes ligados à mesma comunidade. Todas estas variáveis influenciam o desenho e dimensionamento de qualquer proposta de comunicações nesta área.

Assim, temos de referir que o sistema desenvolvido teve em consideração dois cenários possíveis de utilização:

- Ligação de diferentes instituições comunicantes. Neste contexto, o número de máquinas a partilhar informação será eventualmente reduzido mas com grandes volumes de informação por cliente.
- Suporte a ambientes de teletrabalho. Neste contexto, é espectável uma maior quantidade de clientes mas com uma quantidade de informação por cliente muito mais reduzida.

Para qualquer uma das situações, os testes desenvolvidos parecem adequados aos cenários de utilização considerados.

5.2 Trabalho futuro

O módulo desenvolvido foi pensado para responder ao seguinte fluxo de trabalho: pesquisa de informação, selecção de um resultado e transferência de um estudo imagiológico. No entanto, a não é possível transferir dois ficheiros em simultâneo via WAN, isto porque um dos pressupostos considerado no desenvolvimento do módulo foi que cada cliente efectua uma pesquisa de cada vez, o que, em certos cenários, pode não ser verdade. Visto que na plataforma Dicoogle a interface gráfica se encontra desacoplada do núcleo da aplicação, podem existir múltiplos clientes ligados ao mesmo núcleo. Mais ainda, aquando da realização de testes entre múltiplas plataformas Dicoogle verificou-se que existem acessos paralelos a determinados elementos da plataforma. Como referido, tal situação não haviam sido considerados e, por vezes, certas pesquisas podem resultar em excepções devido a acessos concorrenciais. Assim, existe um conjunto de aspectos relacionados

com integração do módulo WAN na plataforma Dicoogle que não foram cobertos e devem ser alvo de atenção em desenvolvimentos futuros.

Outra questão a considerar prende-se com o cliente desenvolvido para a plataforma Dropbox. Para além das aplicações desenvolvidas necessitarem de respeitar a API Dropbox, também necessitam de ser submetidas à aprovação de revisores da Dropbox para poderem passar ao domínio público. Até lá, cada aplicação apenas pode ser usada por um cliente. Assim, a solução desenvolvida para testes e estudo do serviço de armazenamento não pode ser utilizada no dia-a-dia pelos utilizadores do Dicoogle. Recomenda-se pois que se proceda ao processo de registo e certificação do módulo desenvolvido na Dropbox.

No âmbito dos serviços de armazenamento também foi considerado que seria uma mais valia desenvolver modelos estatísticos baseados no histórico dos serviços de armazenamento existentes de forma a optimizar as transferências futuras de informação.

No âmbito da plataforma Dicoogle Mobile desenvolvida, existe todo um conjunto de funcionalidades que a podem aproximar ainda mais da plataforma Dicoogle *desktop* como, por exemplo, a possibilidade de usar os serviços de armazenamento *cloud* que foram estudados nesta dissertação. Também seria interessante desenvolver ferramentas de visualização de imagens DICOM em dispositivos Android. Seria ainda uma mais-valia permitir o registo de relatórios preliminares. Por exemplo, no caso de uma urgência, a plataforma podia permitir a recolha de relatórios de voz, assim com informação sobre o local onde estes foram submetidos.

Uma linha de trabalho mais colateral seria fazer uma análise de custos de operação do serviço *Relay,* usando para isso dados disponibilizados por instituições médicas.

Finalmente, outro aspecto que não foi coberto foi o facto de ser assumido que o serviço se encontra sempre disponível, o que pode não ser verdade. Por exemplo, o acesso ao serviço é vedado se utilizarmos mais recursos do que aqueles que foram contratualizados na plataforma GAE. É assim necessário desenvolver mecanismos que notifiquem o utilizador para situações deste tipo.

6 Bibliografia

[1] P. Sylva, "A situation analysis on pacs prospects for a developing nation," *Sri Lanka Journal of Bio-Medical Informatics*, vol. 1, pp. 112–117, April 2010.

[2] A. Bui, C. Morioka, J. Dionisio, D. Johnson, U. Sinha, S. Ardekani, R. Taira, D. Aberle, S. El-Saden, and H. Kangarloo, "opensourcepacs: An extensible infrastructure for medical image management management," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 11, no. 1, pp. 94 –109, 2007.

[3] G. B. Bell and A. Sethi, "Matching records in national medical patien index," *Commun. ACM*, vol. 44, pp. 83–88, September 2001.

[4] B. Erickson and N. Hangiandreou, "The evolution of electronic image in the medical environment," *Journal of Digital Imaging*, vol. 11, pp. 71–74, 1998. 10.1007/BF03168264.

[5] C. Costa, C. Ferreira, L. Bastião, L. Ribeiro, A. Silva, and J. Oliveira, "Dicoogle - an open source peer-to-peer pacs," *Journal of Digital Imaging*, pp. 1–9, 2010. 10.1007/s10278-010-9347-9.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010.

[7] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, and P. Reynolds, "Cloud computing: A new business paradigm for biomedical information sharing," *Journal of Biomedical Informatics*, vol. 43, no. 2, pp. 342 – 353, 2010.

[8] D. Ogrizovic, B. Svilicic, and E. Tijan, "Open source science clouds," in *MIPRO, 2010 Proceedings of the 33rd International Convention*, pp. 1189 –1192, May 2010.

[9] L. Mei, W. Chan, and T. Tse, "A tale of clouds: Paradigm comparisons and some thoughts on research issues," in *Asia-Pacific Services Computing Conference, 2008. APSCC '08. IEEE*, pp. 464 –469, 2008.

[10] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, (Washington, DC, USA), pp. 23–31, IEEE Computer Society, 2009.

[11] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pp. 4 –16, 2009.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, (Washington, DC, USA), pp. 124–131, IEEE Computer Society, 2009.

[13] J. Varia, “Cloud architectures,” *White Paper of Amazon, jineshvaria. s3. amazonaws. com/public/cloudarchitectures-varia. pdf*, 2008.

[14] K. Keahey, M. Ripeanu, and K. Doering, “Dynamic creation and management of runtime environments in the grid,” in *Workshop on Designing and Building Web Services (to appear)*, Citeseer, 2003.

[15] “Xen.” http://www.xen.org/.

[16] “Kvm.” http://www.linux-kvm.org/page/Main%5C_Page.

[17] “Vmware.” http://www.vmware.com/virtualization/.

[18] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski, “AppScale design and implementation,” 2009.

[19] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM ’09, (Washington, DC, USA), pp. 44–51, IEEE Computer Society, 2009.

[20] G. Zhao, C. Rong, J. Li, F. Zhang, and Y. Tang, “Trusted data sharing over untrusted cloud storage providers,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 30 2010.

[21] “Amazon web services.” http://aws.amazon.com/.

[22] “Nimbus.” http://www.nimbusproject.org/.

[23] “Eucalyptus.” http://www.eucalyptus.com/.

[24] “Windows azure.” http://www.microsoft.com/windowsazure/.

[25] D. Chappell, “Introducing the Windows Azure Platform,” 2010.

[26] “Google app engine.” http://code.google.com/appengine/.

[27] D. Sanderson, *Programming Google app engine*. O’Reilly Media, 2009.

[28] “Salesforce.com.” http://www.salesforce.com/platform/.

[29] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, “Science clouds: Early experiences in cloud computing for scientific applications,” *Cloud computing and applications*, vol. 2008, 2008.

[30] “Appscale.” http://code.google.com/p/appscale/.

[31] J. Lim and R. Zein, “The digital imaging and communications in medicine (dicom): Description, structure and applications,” in *Rapid Prototyping* (H. R. Parsaei, A. Kamrani, and E. A. Nasr, eds.), vol. 6 of *Manufacturing Systems Engineering Series*, pp. 63–86, Springer US, 2006. 10.1007/0-387-23291-5_3.

[32] W. Bidgood, S. Horii, F. Prior, and D. Van Syckle, "Understanding and using DICOM, the data interchange standard for biomedical imaging," *Journal of the American Medical Informatics Association*, vol. 4, no. 3, p. 199, 1997.

[33] "Digital imaging and communications in medicine (dicom) part 1: Introduction and overview," *National Electrical Manufacturers Association*, 2009.

[34] G. Koutelakis and D. Lymberopoulos, "Wada service: An extension of dicom wado service," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 13, no. 1, pp. 121 – 130, 2009.

[35] H. Huang and R. Taira, "Infrastructure design of a picture archiving and communication system," *American Journal of Roentgenology*, vol. 158, no. 4, p. 743, 1992.

[36] "Open source clinical image and object management." http://www.dcm4che.org/.

[37] "Apache sf: Lucene index server." http://lucene.apache.org.

[38] C. Costa, F. Freitas, M. Pereira, A. Silva, and J. Oliveira, "Indexing and retrieving dicom data in disperse and unstructured archives," *International Journal of Computer Assisted Radiology and Surgery*, vol. 4, pp. 71–77, 2009. 10.1007/s11548-008-0269-7.

[39] "Jgroups - a toolkit for reliable multicast communication." http://www.jgroups.org/.

[40] B. Ban, "Reliable Multicasting with the JGroups Toolkit," 2011.

[41] "Best practices for writing scalable applications- google app engine." http://-code.google.com/appengine/articles/scaling/overview.html.

[42] E. Mendonça, E. Chen, P. Stetson, L. McKnight, J. Lei, and J. Cimino, "Approach to mobile information and communication for health care," *international Journal of Medical informatics*, vol. 73, no. 7-8, pp. 631–638, 2004.

[43] K. Hameed, "The application of mobile computing and technology to health care services," *Telemat. Inf.*, vol. 20, pp. 99–106, May 2003.

[44] C. Doukas, T. Pliakas, and I. Maglogiannis, "Mobile healthcare information management utilizing cloud computing and android os," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pp. 1037 –1040, 31 2010-sept. 4 2010.

[45] "Android.com." http://www.android.com/.

[46] S. Khludov, L. Vorwerk, and C. Meinel, "Internet-orientated medical information system for dicom-data transfer, visualization and revision," in *Computer-Based Medical Systems, 2000. CBMS 2000. Proceedings. 13th IEEE Symposium on*, 2000.

[47] V. Ramasubramanian, R. Peterson, and E. Sirer, "Corona: A high performance publish-subscribe system for the world wide web," in *Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pp. 2–2, 2006.

[48] “Channel api overview (java).” http://code.google.com/appengine/docs/java/channel/-overview.html.

[49] “Htmlunit - welcome to htmlunit.” http://htmlunit.sourceforge.net/.

[50] G. Mulligan and D. Gracanin, “A comparison of soap and rest implementations of a service based interaction independence middleware framework,” in *Winter Simulation Conference (WSC), Proceedings of the 2009*, pp. 1423 –1432, dec. 2009.

[51] “Box.net, online file sharing, content management, collaboration.” http://www.box.net/.

[52] “Dropbox online backup, file sync, and sharing made easy.” http://www.dropbox.com/.

[53] “What is android? | android developers.” http://developer.android.com/guide/basics/what-is-android.html.

[54] “Dungeon master on android - project kenai.” http://kenai.com/projects/adungeonmaster.

[55] “Making your own website - html, javascript, php, online marketing and other website making information....” http://www.dom4j.org/.

[56] “kaeppler/dom4j-1.6.1-harmony - github.” http://github.com/kaeppler/dom4j-1.6.1-harmony/downloads.

[57] “Welcome to cloud foundry.” http://www.cloudfoundry.com/.

[58] “Java.net.” http://jersey.java.net/.

[59] “Web application description language.” http://www.w3.org/Submission/wadl/.

[60] “Httpclient - httpcomponents httpclient overview.” http://hc.apache.org/httpcomponents-client-ga/.

[61] “google-gson - a java library to convert json to java objects and vice-versa - google project hosting.” http://code.google.com/p/google-gson/.

[62] “Jaxb reference implementation - java.net.” http://jaxb.java.net/.

Apêndice A Interface REST do serviço de Relay.

Visão geral

De forma a complementar a informação descrita no texto e a fornecer uma especificação completa do sistema desenvolvido, neste apêndice é apresentada uma descrição da API final do serviço de *Relay*[7].

Criar Comunidade

Finalidade	Permite criar uma comunidade.
Pré-condições	-
URL	$(Endereço) /Community
Mensagem	HTTP PUT
Parâmetros	**Parâmetro** / **Descrição** password — Chave que os clientes fornecerão para se conectarem aquela comunidade. defaultKey — Dependendo da configuração do serviço de Relay pode ou não ser necessário fornecer uma palavra-chave do serviço para a criação de uma comunidade.
Tipo Retorno	JSON

Retorno Exemplo	{ "id": "1",
	"password": "palavraChave",
	"type": "standing", //standing Trata-se de uma comunidade permanente.
	//dynamic Apagada na ausência de utilizadores
	}

Login

Finalidade	Permite a um utilizador obter uma entidade do tipo sessão com um *token* de acesso ao sistema.
Pré-condições	-
URL	$(Endereço) /Auth/
Mensagem	HTTP POST
Parâmetros	**Parâmetro** / **Descrição** username — Nickname do cliente que se esta a ligar a uma comunidade. communityID — Identificador da comunidade a que se esta a ligar communityPassword — Palavra-chave para se ligar a comunidade
Tipo Retorno	JSON
Retorno Exemplo	{ "userName": "username", "accessToken": "imak6u8w…j5xa02y", "communityID": "12345", "notifyMode":"0", //0-> Polling Mode, 1->Publish-Subscribe Mode "lastRefresh":"12345", }

[7] Uma vez que os pedidos serão efectuados sobre HTTP as excepções geradas são traduzidas para códigos de erro HTTP pela biblioteca responsável pela interface REST. Assim caso tudo corra bem é devolvido um retorno Status = 200, em caso de falha Status ≠ 200.

Logout

Finalidade	Termina a sessão no serviço de Relay.
Pré-condições	O Utilizador tem de se encontrar com uma sessão activa.
URL	$(Endereço) /Auth/
Mensagem	HTTP DELETE
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso que foi enviado ao cliente durante o processo de *Login*
Tipo Retorno	-

Obter notificações

Finalidade	Permitir a um cliente subscrever as notificações sobre dados a si destinados. Através de polling.
Pré-condições	O cliente deve encontrar-se com uma sessão activa no sistema.
URL	$(Endereço)/PollingClient/
Mensagem	HTTP POST
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*. Body / Lista formata em Json com os identificadores das mensagens a apagar caso existam.
Tipo Retorno	Json
Retorno Exemplo	[…,{"id": "1","content": "{…}",},…]

Subscrever notificações

Finalidade	Permitir a um cliente subscrever as notificações sobre dados a si destinados. Através de publish-subscrive
Pré-condições	O cliente deve encontrar-se com uma sessão activa no sistema.
URL	$(Endereço)/ChannelClient/
Mensagem	HTTP GET
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*.
Tipo Retorno	text/html

Retorno Exemplo	<html>
	<head>
	<script src='/_ah/channel/jsapi'></script>
	</head>
	<body>
	<script type='text/javascript'>
	onOpened = function() {
	alert("onOpened");
	}
	….
	</script>
	</body>
	</html>

Ping

Finalidade	Notificar o serviço de *Relay* que o cliente se encontra activo.
Pré-condições	O cliente deve encontrar-se com uma sessão activa no sistema.
URL	$(Endereço)/Validation/
Mensagem	HTTP POST
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*.
Tipo Retorno	-

Listar clientes activos

Finalidade	Obter a lista de clientes activos na comunidade ao qual o cliente pertence.
Pré-condições	O cliente deve encontrar-se com uma sessão activa no sistema.
URL	$(Endereço)/Community/
Mensagem	HTTP GET
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*.
Tipo Retorno	JSON
Retorno Exemplo	[“usernameX” ,…,”usernameZ”]

Enviar mensagem para muitos

Finalidade	Enviar uma mensagem JSON de um cliente para todos os outros existentes na sua comunidade.
Pré-condições	O cliente deve encontrar-se com uma sessão activa no sistema.
URL	$(Endereço)/ OneToMany/
Mensagem	HTTP PUT
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*. Body / Mensagem JSON a enviar
Tipo Retorno	Text/plain
Retorno Exemplo	“N”: Numero de clientes que deveram receber a mensagem

Enviar mensagem para um

Finalidade	Enviar uma mensagem JSON de um cliente para outro.
Pré-condições	Ambos os clientes origem e destino devem encontrar-se com uma sessão activa.
URL	$(Endereço)/ OneToOne/
Mensagem	HTTP PUT
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*. Body / Mensagem JSON a enviar
Tipo Retorno	Text/plain
Retorno Exemplo	“1”: Indica que devera ser recebida por um único cliente.

Colocar mensagem

Finalidade	Colocar um ficheiro no serviço Relay acessível a outro cliente.
Pré-condições	O cliente origem deve encontrar-se com uma sessão activa.
URL	$(Endereço)/ Storage/
Mensagem	HTTP POST
Parâmetros	**Parâmetro** / **Descrição** accessToken / Token de acesso fornecido ao cliente após o *login*. fileName / Nome que será associado aos dados Body / Dados binários do ficheiro a colocar, Até um 1MB.
Tipo Retorno	JSON
Retorno Exemplo	{ "privateLink": $(Endereço)/Storage/imak6u8w...j5xa02y/QueryResponse.1", "publicLink": "$(Endereço)/Storage/1.usernameX/QueryResponse.1", "storageName": "relayGAE" }

Ler mensagem

Finalidade	Obter os dados de uma mensagem colocada no serviço de *Relay* por outro cliente.
Pré-condições	-
URL	$(Endereço)/ Storage/{source}/{fileName}
Mensagem	HTTP GET
Parâmetros	-
Tipo Retorno	Dados Binários

Apagar mensagem

Finalidade	Apagar os dados de uma mensagem binária previamente colocada no serviço de *Relay*.
Pré-condições	O cliente deve encontrar-se com uma sessão activa.
URL	$(Endereço)/Storage/{accessToken}/{fileName}
Mensagem	HTTP DELETE
Parâmetros	-
Tipo Retorno	-

Apêndice B Bibliotecas

Durante a implementação do módulo de *Relay* foi necessário recorrer ao uso de diversas Bibliotecas Java as quais serão descritas de seguida:

Jersey [58]: Facilita a implementação de interfaces REST, uma importante característica desta biblioteca é a criação uma descrição Web WADL (Web application Description Language) [59] do serviço desenvolvido, este tipo descrição nada mais é que um ficheiro seguindo uma estrutura XML, que pode ser consumido por diversas plataformas de desenvolvimento com o objectivo de criar métodos de acesso para serviço disponibilizado de forma automática.

Httpclient [60]: Biblioteca usada para efectuar os pedidos http do lado do cliente.

HtmlUnit[49]: Foi o browser embutido escolhido para recepção de notificações do GAE channel.

Gson [61]: fornece um conjunto de métodos através dos quais é possível converter a informação de um dado objecto em JSON e vice-versa. Usada para acesso ao serviço de Relay e serviço dropbox.

JAXB [62]: possui funcionalidades tradução de objectos em XML e vice-versa. Usada para acesso aos dados de configuração.