FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Evolução de Correio Electrónico para Modelos de Comunicação em Tempo Real

Héctor Manuel Gomes Dantas

Mestrado Integrado em Engenharia Informática e Computação

Orientador: João Correia Lopes (Prof. Auxiliar)

28 de Julho de 2010

Evolução de Correio Electrónico para Modelos de Comunicação em Tempo Real

Héctor Manuel Gomes Dantas

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: António Fernando Vasconcelos Cunha Castro Coelho (Professor Auxiliar)
Vogal Externo: Maria Benedita Campos Neves Malheiro (Professora Adjunta)
Orientador: João António Correia Lopes (Professor Auxiliar)

---

19 de Julho de 2010

Resumo

Actualmente nota-se uma crescente expansão do número de plataformas de comunicação. O utilizador beneficiaria em ter acesso, na mesma interface, a informação gerada por essas mesmas plataformas. Exemplos dessa comunicação serão *feeds*, mensagens de correio electrónico, *posts* em *blogs*, actividade em redes sociais, entre outros.

Entre os serviços fornecidos pela Portugalmail – Comunicações, S. A., a empresa que propôs este projecto, encontra-se o correio electrónico e o *blog*. O desenvolvimento e gestão de plataformas de correio electrónico é o seu principal negócio. A questão da necessidade e utilidade do correio electrónico é um ponto de interesse para esta empresa e rapidamente se verifica que o utilizador contemporâneo tem algumas necessidades que o correio electrónico convencional não cobre.

Englobando a forte presença do correio electrónico e as novas necessidades dos utilizadores, faz todo o sentido fazer uma convergência entre este e outros métodos de comunicação, nomeadamente comunicação síncrona.

O objectivo deste trabalho será obter uma solução para o problema de juntar a informação de várias fontes e apresentá-las ao utilizador na mesma interface, permitindo ainda que este interaja com estas fontes nesta mesma aplicação. A solução passará por uma aplicação cliente/servidor e o protocolo usado para a comunicação será o *Extensible Messaging and Presence Protocol* (XMPP). A solução permitirá a implementação deste produto através da adaptação de tecnologia *open-source* existente.

Depois de definida a abordagem é implementado um protótipo de maneira a comprovar a proposta de solução. Apesar de não ter sido provada a escalabilidade do protótipo, foram tomadas decisões ao longo da sua concepção para este ser o mais escalável possível. Estas decisões foram baseadas num estudo feito ao elemento nuclear do sistema: o servidor XMPP. Baseado nestes testes, foi escolhido o servidor Ejabberd. Finalmente, a integração com a plataforma de correio electrónico e gestor de contactos da empresa foi alcançada.

Abstract

Today we see the growing expansion of the number of media platforms. The user benefits from having access, on the same interface, to information generated by all these platforms. Examples of such communication are feeds, emails, posts on blogs, social networking activity, among others.

Among the services provided by Portugalmail – Comunicações, S. A., the company that initially proposed this project, are the e-mail and blog. The development and management of email platforms is their primary business. The question of the necessity and usefulness of email is a point of interest for this company. The contemporary user has some needs that conventional e-mail does not cover.

When thinking of the strong presence of electronic mail and the new needs of users, it makes sense to do a convergence between email and other methods of communication, including synchronous communication.

The work objective is to obtain a solution to the problem of gathering information from various sources and present it to the same user interface and allowing interaction with these sources in the same application. The solution is a client/server application and the protocol used for communication is the Extensible Messaging and Presence Protocol (XMPP). The solution will enable the implementation of this product by adapting existing open-source technology.

After defining the approach, a prototype is implemented to demonstrate how the proposed solution works and serves as a proof of concept. Despite not having been proven scalability of the prototype, some decisions were made over its conception for this to be the most scalable as possible. These decisions were based on a benchmark done to the main element of the system that is the XMPP server, based on these tests was chosen ejabberd. Finally, integration with the company's e-mail platform and contact management platform was reached.

Agradecimentos

Quero expressar os mais sinceros agradecimentos à instituição FEUP, em particular ao MIEIC, na figura do seu director o Professor Augusto Sousa. Estou especialmente agradecido ao Professor João Correia Lopes pelo apoio e valor que acrescentou a esta dissertação.

Quero expressar a minha gratidão a todos na Portugalmail por tornarem este estágio possível e me fazerem sentir sempre bem-vindo e à vontade. Gostaria de destacar o Nuno Lopes pela orientação e suporte que me deu ao longo do projecto.

Não podia deixar de destacar o papel da minha família e dos amigos pelo apoio moral, amor e carinho.

Finalmente, e rapidamente uma vez que as minhas palavras são parcas, gostaria de agradecer a todas as pessoas que me ofereceram a sua ajuda, apoio e paciência.

Héctor Manuel Gomes Dantas

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ACL	Access Control List
AIM	AOL Instant Messenger
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
BOSH	Bidirectional-streams Over Synchronous HTTP
CMS	Content Management System
CSS	Cascading Style Sheets
ESMT	European School of Management and Technology
GPL	General Public License
GWT	Google Web Toolkit
HTML	HyperText Markup Language
ICQ	I Seek You
IDE	Integrated Development Environment
IETF	The Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IRC	Internet Relay Chat
IM	Instant Messaging
JID	Jabber ID
JSON	JavaScript Object Notation
LGPL	Lesser General Public License
MIT	Massachusetts Institute of Technology
MSN	Microsoft Service Network
PHP	PHP: Hypertext Preprocessor
POP3	Post Office Protocol
REST	Representational State Transfer
RDF	Resource Description Framework
RFC	Request For Comments
RSS	Really Simple Syndication
SGBD	Sistema de Gestão de Base de Dados
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
XEP	XMPP Extension Protocols
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XSF	XMPP Standards Foundation

Capítulo 1

Introdução

Durante o ano de 2009 foram lançadas ferramentas que almejam tornar a comunicação dos utilizadores da Internet mais síncrona. Levantou-se, novamente, a questão da necessidade e utilidade do correio electrónico enquanto ferramenta de comunicação assíncrona que é. O objectivo destas novas ferramentas é agregar métodos de comunicação provenientes de várias fontes e tornar esta informação rapidamente disponível ao utilizador, fazendo com que esta tenda para o acesso instantâneo. Estas ferramentas além de agregarem a informação e a tornarem disponível para o utilizador no mesmo local, também possibilitam a interacção entre o utilizador e as fontes de comunicação através de comentários ou respostas.

Um estudo [Per09] revelou que as redes sociais, como é o caso do Facebook, do MySpace ou do LinkedIn, estão a contribuir para o declínio do correio electrónico e dos *instant messengers*. Neste estudo é argumentado que apesar de o correio electrónico, no ambiente empresarial, ainda ser essencial, em termos pessoais e de lazer as redes sociais fornecem uma forma mais eficaz de comunicação. Ou seja, existem necessidades do utilizador contemporâneo que não são contempladas pelo correio electrónico.

1.1 Enquadramento

A evolução das ligações de Internet para a possibilidade de o utilizador estar constantemente ligado em rede levou ao aparecimento de um tipo de comunicação mais síncrona e menos formal. O utilizador para saber se alguém já teria saído do trabalho não necessitaria de lhe mandar uma mensagem e esperar pela resposta e simplesmente teria de ver o seu estado em plataformas como o Facebook ou Twitter ou falar num *instant messenger* com a pessoa em questão. Para partilhar as fotos das férias não necessitaria de enviar uma mensagem com anexos a cada um dos seus contactos bastando para isto partilhá-la no seu *blog* ou Facebook [Vas10]. Tratam-se de alguns

exemplos da necessidade que o utilizador tem, na actualidade, de estar constantemente e instantaneamente actualizado. A informação encontra-se dispersa por várias plataformas e localizações, o que dificulta a visão global pelo utilizador.

Este trabalho enquadra-se no ambiente empresarial Portugalmail – Comunicações, S. A. [PC10f]. Esta Empresa actua na área das novas tecnologias, nomeadamente desenvolvendo soluções integradas de gestão e comunicação, dedicando-se ao fornecimento de serviços de presença e comunicação na Internet [PC10g].

Entre os serviços fornecidos pela Portugalmail – Comunicações, S. A. encontra-se o correio electrónico e o *blog*. O desenvolvimento e gestão de plataformas de correio electrónico é o seu principal negócio. Entre as soluções de correio electrónico encontra-se o Email Gratuito [PC10c], uma solução disponível para qualquer utilizador e que é rentabilizado pela publicidade apresentada; o Bussiness Email [PC10b] que é uma versão de correio electrónico para empresas e o Portal Email ASP [PC10e] dirigido a portais e empresas de telecomunicações que necessitem de um serviço de correio electrónico e optem por fazer *outsourcing* em alternativa ao desenvolvimento de uma solução própria. Tal como será explicado mais à frente neste capítulo, estas soluções são importantes na justificação do desenvolvimento deste trabalho. O outro serviço enunciado, o *blog*, possui duas soluções: o Blog Gratuito [PC10a], que é uma solução gratuita e disponível a qualquer utilizador, e a Portal Blog ASP [PC10d], que é uma solução que permite a qualquer empresa a criação do seu próprio serviço de alojamento de *blogs*.

Como se pode verificar, o universo da Portugalmail – Comunicações, S. A. baseia-se na comunicação desde a troca de mensagens directas, no caso do correio electrónico, até à difusão de informação dos *blogs*. No entanto, estes tipos de comunicação são assíncronos [RRGC06], ou seja, o receptor não recebe nem é alertado em tempo real de informação nova. Nesta dissertação é enunciada inúmeras vezes a expressão: tempo real; que não deve ser levada à letra, pretende-se transmitir a ideia de comunicação em tempo útil.

1.2 Motivação

Face ao enquadramento exposto, rapidamente se verifica que o utilizador contemporâneo tem algumas necessidades que o correio electrónico convencional não cobre.

Devido à crescente expansão do número de plataformas de comunicação, o utilizador beneficiaria em ter acesso, na mesma interface, à informação gerada por essas mesmas plataformas. Exemplos dessa informação serão mensagens de correio electrónico, *posts* em *blogs*, actividade em redes sociais, entre outros.

Mais do que permitir unicamente a visualização desta informação, também seria benéfico o utilizador poder usar a aplicação para interagir com estas fontes. Por exemplo, ao receber o *post* de um amigo no seu *blog*, poder deixar imediatamente um comentário sem ser necessário visitar o *blog*. Neste caso, verificam-se duas funcionalidades que o utilizador cada vez mais procura: primeiro, é notificado imediatamente de nova informação do seu interesse e, segundo, tem a possibilidade de interagir rapidamente, sem necessitar de navegar para outras plataformas.

Como exemplificado no parágrafo anterior, o utilizador da *Web* contemporâneo procura estar actualizado [Sie09] e para tal precisa de receber informação do seu interesse de uma forma o mais imediata e célere possível. Não adianta um utilizador efectuar um *post* no seu perfil do Twitter a dizer que acabou de sair do emprego se outro utilizador só verificar esse *post* no dia seguinte. Por conseguinte, torna-se óbvio que cada vez mais o utilizador precisa de ter acesso à informação, o mais rapidamente possível.

Face às novas ferramentas introduzidas com o objectivo de substituir o correio electrónico transformando as comunicações assíncronas em síncronas, torna-se necessário introduzir também a capacidade de efectuar comunicações síncronas em serviços que dispõem unicamente de comunicação assíncrona como é o caso do correio electrónico disponibilizados pela Portugalmail.

No entanto, apesar de estar constantemente a ser profetizada a sua morte, o correio electrónico continua a ser a maneira preferencial de comunicar na *Web* por parte dos utilizadores e as novas ferramentas que vão surgindo servem apenas de complemento. Aliás, o correio electrónico tem vindo a verificar um aumento da sua utilização e apesar de este crescimento ser em percentagem inferior ao das novas plataformas, ele continua a ser indiscutivelmente o mais utilizado [Sch09].

Devido a esta forte presença do correio electrónico e destas necessidades dos utilizadores, faz todo o sentido fazer uma convergência entre este e outros métodos de comunicação, nomeadamente comunicação síncrona. Concluindo, a agregação de informação de várias fontes e a sua visualização e interacção com rapidez justifica-se na actualidade.

1.3 Objectivos

O objectivo principal do trabalho, tal como está explícito no seu título é a evolução do correio electrónico para modelos de comunicação em tempo real. Ou seja, o objectivo é obter uma solução para o problema de juntar informação de várias fontes, incluindo do correio electrónico, e apresentá-la ao utilizador na mesma interface, sendo possível que este interaja com estas fontes numa só aplicação. Este fluxo de informação deverá ocorrer o mais próximo possível da comunicação em tempo real.

Para este efeito será implementada uma aplicação composta por um servidor e respectivo cliente. O cliente será desenvolvido em JavaScript uma vez que se pretende que este seja uma aplicação para ser acedida através de um *browser*, nomeadamente a partir da interface do correio electrónico da Portugalmail. Permitir visualizar de uma forma apelativa toda a informação agregada, nomeadamente informação multimédia, encontra-se entre os objectivos deste *front end*. É esperado também que a aplicação tenha uma elevada usabilidade com a inclusão de JavaScript. A informação será apresentada agrupada por utilizador (informação das várias contas do mesmo utilizador) e agrupada por plataforma (por exemplo, *posts* do Twitter de contas que não estejam relacionadas com outros contactos).

O servidor será desenvolvido para poder correr em Linux e terá uma base de dados. Uma das funções deste servidor será recolher informação das várias fontes através das suas Application Programming Interfaces (API). Esta recolha será efectuada com uma determinada frequência para

todos os utilizadores activos no sistema. Após receber a informação, o servidor irá reencaminhá-la para os utilizadores devidos. Outra das funções será receber mensagens provenientes dos utilizadores e reencaminhá-las para as plataformas indicadas nessa mensagem realizando a acção indicada na mensagem, por exemplo, publicar uma resposta no Twitter.

A base de dados nesse servidor servirá para guardar a informação pessoal de cada utilizador. Também irá guardar os seus contactos e informações necessárias para a comunicação com as API.

O protocolo usado para a comunicação entre cliente e servidor será o Extensible Messaging and Presence Protocol (XMPP) [SM05]. O objectivo e razão de usar este protocolo deve-se ao facto de ser um protocolo aberto e extensível, baseado em Extensible Markup Language (XML), para sistemas de mensagens instantâneas que poderá ser adaptado para as necessidades do sistema. O XMPP é um protocolo bastante difundido, usado em empresas como a Google e a Sun, o que origina uma maior confiança na sua utilização.

Este trabalho tem por objectivo encontrar uma solução para o problema apresentado e, partindo desta, desenvolver um projecto de raiz ou através da adaptação de tecnologia *open-source* existente. Numa perspectiva mais global, esta solução dará uma resposta às novas necessidades dos utilizadores nomeadamente os clientes da Portugalmail – Comunicações, S. A. de forma a permitir valorizar o produto já disponibilizado pela empresa.

1.4 Estrutura do documento

Além da introdução, este documento tem mais sete capítulos. O capítulo 2 apresenta o estudo do estado da arte, apresentando aplicações que abrangem alguns dos objectivos. Uma vez que o protocolo usado é o XMPP, o capítulo 3 faz uma introdução a este protocolo e a tecnologias que estarão presentes na elaboração do protótipo. No capítulo 4 apresenta-se a proposta elaborada para abordar os objectivos definidos na introdução, que originaram os requisitos também apresentados neste capítulo. No capítulo 5 faz-se o processo de escolha das aplicações para o arranque do protótipo. É também abordada a junção de todas as aplicações e configurações entre estas. No capítulo 6 expõe-se a elaboração do protótipo, implementação das funcionalidades especificadas na proposta de solução. O capítulo 7 aborda a integração do protótipo com a plataforma de correio electrónico existente na Portugalmail. Finalmente, o capítulo 8 são apresentadas as conclusões finais da dissertação e os trabalhos futuros.

Capítulo 2

Agregadores de Informação Dispersa com Visualização e Interacção em Tempo Real

Neste capítulo faz-se o estudo do estado da arte. As duas primeiras secções apresentam duas aplicações que abrangem grande parte dos objectivos da dissertação e a terceira secção apresenta de uma forma mais global algumas aplicações que disponibilizam partes dos objectivos.

2.1 Introdução

Neste capitulo apresentam-se algumas ferramentas similares ou que realizam algumas funcionalidades desejadas neste trabalho, ficando assim introduzido o contexto e permitindo um conhecimento mais profundo do projecto e das suas restrições. A ferramenta Raindrop da Mozilla funciona como agregador e permite a interacção com as fontes. O produto da Google com o nome de Wave tem como funcionalidade nuclear a comunicação em tempo real, assim como a agregação de informação de várias fontes. Por fim os agregadores, como o nome sugere, fornecem funcionalidades para agregar informação de várias fontes. As tecnologias e aplicações aqui referidas são as mais relevantes para o projecto que se descobriram durante a investigação, embora no universo de agregação existam inúmeras ferramentas que não são mencionadas.

2.2 Mozilla Raindrop

Nesta secção é apresentada a ferramenta Raindrop e os pontos nos quais se relaciona com este trabalho. São também apresentados alguns detalhes da implementação desta ferramenta.

2.2.1 Introdução

A comunicação *online* actual divide-se em inúmeros serviços de mensagens que se ignoram uns aos outros e que usam padrões e protocolos diferenciados.

Existem canais isolados que exigem que os utilizadores se autentiquem em várias aplicações e serviços *Web* para obter as novas mensagens. Com o objectivo de resolver este problema, o Raindrop foi desenvolvido para funcionar como uma "caixa de entrada universal", agregando correio electrónico, mensagens instantâneas e um conjunto de mensagens de *sites* específicos.

O ponto de honra do Raindrop é o facto deste organizar todos os tipos de comunicação do utilizador de forma a se verificar um acesso intuitivo e fácil a comunicações pessoais separando-as de comunicações de menor interesse. O filtro do Raindrop separa automaticamente conversas individuais de discussões em grupo e mensagens pessoais de anúncios automatizados e actualizações. O algoritmo de filtragem parece entender que, para o utilizador, uma mensagem que chega é ou não importante com base em quem a enviou e no seu conteúdo, não importando o seu *site* ou protocolo.

O Raindrop está actualmente a ser desenvolvido pela Mozilla Labs e ainda é um protótipo disponível para ser obtido para experimentar e desenvolver.

2.2.2 Funcionamento

O Raindrop é um sistema cliente/servidor. O servidor armazena e permite o acesso à informação de cada utilizador. O cliente funciona sobre um navegador *Web* com uma interface para as funcionalidades da ferramenta.

Uma vez que tem a funcionalidade de agregador de mensagens, esta aplicação tem como objectivo recolher automaticamente as mensagens, notificações e comentários de múltiplas contas de correio electrónico e *sites* como *blogs* e redes sociais. O utilizador apenas necessita de fornecer os dados necessários para obter essa informação (dados de autenticação e outros) e ela é agrupada sob a mesma interface.

O Raindrop faz uma tentativa de separar mensagens às quais o utilizador deve responder em contraste com as que se tratam unicamente de informação, também permitindo fazer *clusters* de comunicações, categorizando-as. As mensagens importantes são evidenciadas na interface enquanto que as outras ficam mais em segundo plano. Para além destas duas categorias de mensagens, existe o conceito de *clusters* que agrega, por exemplo, mensagens de grupos de discussão. Relativamente a *microblogging*, mais especificamente o Twitter, o Raindrop classifica mensagens directas e respostas a *tweets* (identificados pelo @) como mais importantes que os avisos gerais de *status* e agrupa as mensagens que pertencem ao mesmo diálogo.

Um dos pontos pretendidos pela equipa de desenvolvimento é permitir responder às comunicações de uma maneira intuitiva e sem abandonar a ferramenta. Dependendo da origem da mensagem esta resposta poderá ser facilitada ou não.

As mensagens recebidas no Raindrop são sujeitas a um tratamento com o intuito de as tornar mais auto-contidas. Por exemplo, os *links* de páginas como o FlickR e o Youtube, inseridos na

mensagem, são substituídos por *tumbnails*. No caso dos vídeos, carregando no seu *tumbnail*, é possível a sua visualização na mesma interface, sem ser necessário mudar de aplicação.

2.2.3 Detalhes de Implementação

O *back end* (servidor) da aplicação é uma base de dados não relacional com o nome de CouchDB. As componentes responsáveis por processar e recuperar mensagens são codificadas em Python dando uso à *framework* Twisted Networking [Moz10].

O *front end* (cliente) é implementado em HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript através da Dojo JavaScript Library [Moz10].

O CouchDB armazena os documentos no formato JavaScript Object Notation (JSON) e tem uma API Representational State Transfer (REST). O HTML, CSS e JavaScript são armazenados na base de dados e são servidos ao utilizador através do servidor interno do CouchDB.

No anexo A encontra-se o esquema da arquitectura do Raindrop.

2.2.4 Conclusão

O Raindrop aborda o mesmo problema que este trabalho se propõe solucionar embora tenha a funcionalidade de filtros e agrupamento de mensagens que não são contempladas nesta dissertação. O Raindrop ainda não contempla alguns pontos do trabalho como o agrupamento de contactos nem, evidentemente, a integração com as plataformas da Portugalmail.

2.3 Google Wave

Nesta secção é apresentada a ferramenta Wave e os pontos nos quais se relaciona com a dissertação. São também apresentados alguns detalhes da implementação da ferramenta.

2.3.1 Introdução

O Google Wave é uma plataforma de comunicação e colaboração [Fer09]. O Wave incorpora várias tecnologias *Web* como correio electrónico, *Instant Messaging*, *wiki*, documentos *online*, redes sociais e gestão de projectos.

O Google Wave pode ser descrito como um conjunto de três camadas [Fer09]: a camada do produto, a camada da plataforma e a camada do protocolo.

2.3.2 Camada do Produto

Esta camada representa o cliente do Google Wave. É uma aplicação *Web* que pode ser usada para aceder e editar *waves*. Uma *wave* é um contentor onde se desenvolve uma comunicação que pode passar por texto, vídeo ou imagens.

É uma aplicação desenvolvida com a *framework* GWT (Google Web Toolkit), que tem como base HTML e JavaScript/AJAX para criar uma interface com uma usabilidade do tipo *desktop*.

A aplicação tem uma taxa de actualização que se aproxima do tempo real chegando ao pormenor de ser possível visualizar o que um utilizador está a escrever, letra a letra. É possível agregar informação de uma fonte exterior, como por exemplo o Twitter, numa *wave* e implementar funcionalidades de interacção com recurso à sua API. As *waves* têm funcionalidades como *spellchecker* e mesmo tradutor em tempo real.

A Google dá ênfase à capacidade do Wave para funcionar como uma plataforma de trabalho colaborativo em tempo real. Os utilizadores podem criar documentos em colaboração e debater entre si enquanto o fazem.

A visualização das conversas na *wave*, pode evoluir no formato de árvore. É possível, ao criador da *wave*, definir que utilizadores têm acesso a cada ramo da árvore. O cliente Wave possui uma função de *playback* para cada uma das *waves* individuais, o que permite visualizar a construção da conversa do inicio ao fim como se de um filme se tratasse. Além do *spellchecker* o Google Wave faz *sintaxcheck*. Linguagem Natural, como é conhecida na inteligência artificial, é usada para corrigir o *input* dos utilizadores. Outra das inovações do Wave é a capacidade de partilhar ficheiros com um simples *drag-and-drop*. Apenas arrastando o ficheiro para dentro da aplicação e os utilizadores com acesso à *wave* obtêm imediatamente acesso ao ficheiro.

Em conclusão, esta camada é a interface com os utilizadores do sistema uma vez que apresenta as funcionalidades ao utilizador.

2.3.3 Camada da Plataforma

A Google Wave disponibiliza APIs abertas que permitem que os utilizadores aumentem as funcionalidades do cliente e também usem as funcionalidades além de ser na interface do Wave. Existem as Wave Extensions (extensões) que são adicionadas a *waves* e aumentam assim o seu leque de opções ou funções e existem as Embedded Wave (*waves* embutidas).

Uma extensão é uma aplicação que funciona dentro de uma wave. As extensões podem ser de dois tipos: Gadgets ou Robots. Uma *gadget* é uma aplicação que pode ser interactiva com o utilizador; um exemplo de uma *gadget* é a Twittergadget que permite submeter e visualizar as mensagens do Twitter dentro de uma *wave*. Um *robot* é um utilizador que se adiciona às *waves* com o intuito de realizarem acções sobre a *wave*, um exemplo é o *robot* de tradução que paralelamente à escrita do utilizador vai efectuando a tradução na *wave*. As extensões podem adicionar desde funcionalidades úteis a funcionalidades lúdicas e podem ser implementadas por qualquer programador e adicionadas ao Wave [Goo10b].

Embbeded Waves é um método de adicionar *waves* a sites externos. Incluídas com as *waves* são adicionadas todas as funcionalidades permitidas numa *wave*.

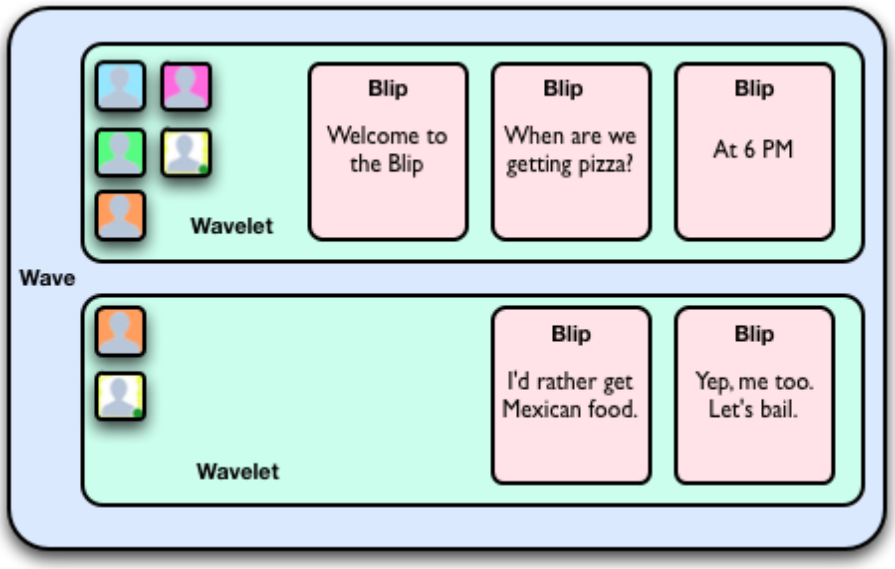


Figura 2.1: Entidades Wave.

2.3.4 Camada do Protocolo

O Google Wave Protocol especifica as regras de formato de base de armazenamento e os meios de partilha de *waves* e inclui o controlo de concorrência, que permite que edições se reflictam de imediato através de utilizadores e serviços. Para incentivar a adopção do protocolo, o Google disponibilizou grande parte do código fonte do Google Wave. O Google Wave Protocol, protocolo usado pelo Google Wave, é baseado no XMPP.

Uma especificação chave do protocolo é que uma *Wave*, em comparação com um *mail*, é um objecto que não é enviado de utilizador para utilizador mas que permanece uma única instância num servidor onde pode ser acedida e editada. Esta *Wave* é um documento XML dinâmico que é construído e editado ao longo do tempo ao contrário da ideia estável que temos dos documentos XML. Uma *wave* tem os constituintes que se pode ver na Figura 2.1[1].

Uma *Wave*, especificamente, refere-se a uma conversa encadeada. Ela pode incluir apenas uma pessoa ou pode incluir um grupo de utilizadores ou até mesmo robôs. A *Wavelet* é também uma conversa encadeada, mas contém apenas um subconjunto de uma conversação maior (uma *Wave*). Ainda menor do que uma *Wavelet*, um *Blip* é uma mensagem única e individual. Cada *Wave* tem um conjunto de um ou mais participantes humanos ou Robôs que interagem numa Wave. Os participantes são adicionados a uma *Wave* por participantes existentes [Goo10a].

O Google Wave inclui um protocolo em código aberto para ser possível a partilha de *Waves* entre os diferentes servidores, utilizando uma abordagem federada. Essencialmente, isso significa que qualquer um pode configurar um servidor que utilize o Google Wave Federation Protocol [LT10].

[1] Fonte: http://code.google.com/apis/wave/guide.html

2.3.5 Conclusão

O Google Wave é uma ferramenta bastante completa apesar de ainda ser numa versão de testes. O Google Wave Protocol é baseado em XMPP e implementa várias funcionalidades interessantes e que se podem comparar às necessárias para este projecto. De notar que esta aplicação não permite agrupar e integrar mensagens e informação de plataformas diferentes, ou seja, a informação permanece separada por plataforma. A noção de evolução do correio electrónico desta abordagem passa pela substituição da mensagem de *mail* pelos diferentes tipos de mensagens disponíveis (chamadas *Waves* e com características diferentes das do *email*).

2.4 Digsby

Esta secção aborda algumas das funcionalidades da ferramenta Digbsy que estão mais relacionadas com esta dissertação. É apresentada uma análise onde são evidenciados pontos que a dissertação aborda, mas a ferramenta não tem soluções implementadas.

2.4.1 Introdução

O Digsby é uma aplicação de *desktop* para Microsoft Windows. A sua principal função é servir como uma aplicação de *instant messaging*, mas possui inúmeras funcionalidades que a completam como ferramenta de agregação de informação e possibilita mesmo a interacção com essa informação.

2.4.2 Funcionalidades

O Digsby permite uma lista combinada de todos os contactos das contas de *instant messaging* do AOL Instant Messenger (AIM), Microsoft Service Network (MSN), Yahoo, I Seek You (ICQ), Google Talk, Jabber e Facebook e permite agrupar estes contactos de maneira a não existirem contactos duplicados.

Relativo a correio electrónico o Digsby permite a introdução de contas de Internet Message Access Protocol (IMAP), Post Office Protocol (POP), Gmail, Hotmail entre outras. À medida que os *emails* chegam são enviadas notificações ao utilizador e, carregando nessas notificações, o utilizador é reencaminhado para a plataforma *Web* dessa conta com a autenticação efectuada. A aplicação também permite o envio de *emails* directamente da janela de *instant messaging*.

Respeitante a outras plataformas, como redes sociais e *microblogging*, o Digsby permite que o utilizador receba actualizações da sua conta do Facebook, Twitter, MySpace e LinkedIn. A informação de cada conta aparece numa janela com a lista de actualizações, sendo possível enviar mensagens e comentários de resposta. Actualizando o estado no Digbsy é possível automaticamente actualizar o estado de cada uma das contas.

2.4.3 Conclusão

O Digsby é uma aplicação bastante completa e abrange bastantes dos objectivos propostos, mas relativamente ao correio electrónico funciona apenas como um notificador e uma ferramenta de envio de *emails*.

Apesar de permitir a interacção com outras plataformas, com grande usabilidade, a experiência dada ao utilizador poderia ser melhor se fosse evidenciada a informação mais importante.

2.5 Outros Agregadores de Informação

Nesta secção é apresentado o funcionamentos dos agregadores de *feeds*. É evidenciado um agregador de *feeds* devido às suas funcionalidades.

2.5.1 Introdução

Um agregador é um cliente de *software* ou aplicação *Web*, que colecciona conteúdo disponível na Internet. Este conteúdo pode ser desde *feeds* Really Simple Syndication (RSS) a informação extraída com recurso a diversas API. O agregador disponibiliza esta informação toda na mesma interface, possivelmente com a utilização de filtros.

Esta aplicação justifica-se para quem tem várias contas em plataformas de meios de comunicação ou redes sociais e partilha informação importante. Estes utilizadores que pretendem partilhar toda esta informação na mesma aplicação podem usar esta tecnologia. Também se justifica para quem quer seguir a informação de várias fontes no mesmo sítio, poupando assim a deslocação por várias plataformas.

2.5.2 Funcionamento

Os agregadores quando ligados a uma fonte de informação verificam se existe novo conteúdo e recuperam-no em intervalos definíveis pelo utilizador.

A tecnologia é conhecida por Pull Technology, já que o agregador é que puxa as novidades do servidor onde estão alojados os conteúdos. Em comparação, o correio electrónico e o IM usam uma metodologia conhecida como Push Tecnhology onde os interveniente “empurram” a informação [Wig09].

O conteúdo distribuído que um agregador irá recuperar e interpretar é normalmente fornecido em forma de RSS, XML ou JSON e noutros dados formatados, tais como Resource Description Framework (RDF)/XML ou Atom.

2.5.3 Friendfeed

O FriendFeed é um agregador *Web* com um grande leque de fontes possíveis. Entre estas fontes encontram-se actualizações de redes sociais, meios de comunicação social, *blogs*, *microblogs* e *bookmarking* social assim como as comuns *feeds*.

O FriendFeed permite subscrever as listas de outros utilizadores que se pretendam seguir. Este outro conceito que o FriendFeed desenvolve é a interacção social: para além de seguirmos as listas dos amigos, é possível comentar os elementos desta lista. Estes comentários ficam exclusivamente no FriendFeed e não são enviados para as fontes.

O FriendFeed suporta mais de 50 diferentes fontes de informação. As mais populares entre elas são YouTube, Facebook, Digg, Google Reader, Twitter, Flickr, StumbleUpon, Tumblr e Last.fm.

2.5.4 Conclusão

Existem, na actualidade, inúmeros agregadores como o Google Reader, o Bloglines e outros. Quando se pretende seguir inúmeras fontes de informação na *Web* a melhor forma de se manter actualizado é utilizar um agregador, uma vez que este apresenta nova informação quando ela é criada e assim o utilizador já não precisa de despender tempo a visitar cada fonte. O próximo passo lógico, e que os agregadores ainda não permitem, é interagir com essas mesmas fontes.

2.6 Conclusão

Existem várias ferramentas em desenvolvimento por grandes companhias que pretendem obter os mesmos resultados, apesar de ser em contextos diferentes daquele que este projecto se insere. Apesar de haver uma forte tentativa de aglomeração de informação e tentativa de agilizar a comunicação com as fontes de informação, o ponto fulcral, que é a tentativa de evoluir o correio electrónico para modelos em tempo real, apenas é abordado por duas ferramentas estudadas (Raindrop e Wave).

Capítulo 3

Tecnologias XMPP para Agregação de Informação e Comunicação em Tempo Real

Neste capítulo é apresentada uma breve descrição do funcionamento do XMPP uma vez que é o protocolo que se pretende usar no projecto. São apresentadas também algumas tecnologias (XMPP e outras) que poderão ser usadas no projecto, incluindo as suas características actuais e a forma como estas poderão influenciar o desenvolvimento da solução proposta.

3.1 Protocolo XMPP

Nesta secção são apresentadas as características do protocolo XMPP assim como pormenores da sua arquitectura. São também apresentados os detalhes mais relevantes para perceber o seu funcionamento.

3.1.1 Introdução

O XMPP é um protocolo de comunicação em tempo real e de informação de presença baseado em XML, completamente livre, tornando-o tão extensível que já existem quase 250 extensões oficiais ao protocolo. Informação de presença é um indicador de disponibilidade de um utilizador para comunicar: um cliente envia a informação de presença de um utilizador para um servidor indicando que este está disponível. É possível a qualquer pessoa com uma ligação à Internet e um domínio criar um servidor próprio e comunicar com pessoas noutros servidores.

A origem do XMPP remonta a 1998, altura em que Jeremie Miller começou a escrever as primeiras linhas do protocolo que foi desenvolvido pela Jabber Open-source Community em 1999 e formalizado pela IETF em 2002-2004. O protocolo inicialmente era denominado Jabber, passando a designar-se XMPP na altura que foi formalizado (ainda se usa muito Jabber para se referir ao

protocolo XMPP). É continuamente estendido através do processo de normas da XMPP Standards Foundation (XSF) [Fou10c].

A flexibilidade é um ponto forte do XMPP. É possível controlar uma rede, jogar, fazer *chats* para pessoas que não estão na nossa lista de contactos, entre outros.

3.1.2 Arquitectura XMPP

Na arquitectura do XMPP o cliente tem um identificador único (JID) e comunica com outro cliente através de um servidor. Se este cliente estiver situado noutro domínio, o servidor comunica com o servidor que disponibiliza esse domínio, como se pode verificar na Figura B.1 no Anexo B que se encontra na página 70. O servidor presta o serviço de encaminhamento entre domínios. Os JID ou Jabber ID incluem o domínio, o nó (opcional), o recurso (opcional) e são constituídos da seguinte forma [Jon09]:

```
[ node "@" ] domain [ "/" resource ]
```

Podem existir transportes (*Gateways/Transports*), como se pode ver na Figura B.2 no Anexo B que se encontra na página 70, com o objectivo de permitir a ligação a servidores com outros protocolos e que estarão a correr uma extensão do XMPP. Os contactos que estejam nos servidores de outros protocolos (MSN, ICQ, AIM, *etc*.) serão então incorporados na nossa lista de contactos. Este processo provoca uma diminuição no desempenho, porque envolve mais um servidor no meio (a *gateway*). O XMPP serve de protocolo de suporte para fornecer conectividade universal entre os diferentes protocolos [Jon09].

Ao contrário de outros protocolos, é possível ter uma mesma conta ligada em clientes diferentes. Para tal, basta atribuir-lhe um recurso (exemplo: exemplo@jabber.com/Home, exemplo@jabber.com/Mobile, exemplo@jabber.com/work, *etc*.). Paralelamente, o utilizador atribui uma prioridade numérica a cada um dos recursos. Quando o servidor jabber.com receber uma mensagem destinada ao utilizador "exemplo", irá encaminhá-la para o recurso que tiver maior prioridade.

3.1.3 O Núcleo do XMPP

Uma *stream* XML é um contentor que serve como apoio para a troca de informação XML entre duas entidades. Esta informação tem como elemento raiz a *tag* <stream/> que é construída ao longo do tempo com *stanzas* XML. As *stanzas*, que são o primeiro filho do elemento <stream/>, são usadas no XMPP para comunicar mensagens ou informação de presença [SM05]. Os três tipo de *stanzas* existentes são as seguintes:

- A *stanza* <message/> é um mecanismo pelo qual uma entidade envia informações para outra entidade, semelhante às comunicações que ocorrem num sistema como *e-mail*;
- A *stanza* <presence/> é um mecanismo de *broadcast* através do qual várias entidades recebem informações sobre a entidade que subscreveram (em particular, a informação sobre a disponibilidade de uma entidade de rede);

- A *stanza* <iq/> (Info/Query) é um mecanismo de pedido-resposta semelhante ao HTTP, que permite a uma entidade fazer um pedido e receber uma resposta de outra entidade.

O XMPP é um protocolo que se processa sobre *sockets* TCP. Com base nas *streams* XML, as *stanzas* são transferidas assincronamente. Na Figura B.3 no Anexo B que se encontra na página 71 pode ver-se um exemplo da construção de uma *stream* durante uma comunicação de IM [Jon09].

Um erro é devolvido da seguinte forma:

```
<stream:error>
  <xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
<stream:error>
```

Estas especificações núcleo do XMPP estão mencionadas nos Request for Comments (RFC) 3920 e 3921 [Fou10b]. A comunidade XMPP continua a expandir o protocolo através de XMPP Extension Proposals (XEP).

3.1.4 Conclusão

O XMPP é um protocolo aberto, extensível, baseado em XML, desenvolvido originalmente para mensagens instantâneas e informação de presença formalizado pelo Internet Engineering Task Force (IETF). A arquitectura da rede XMPP é similar à do *e-mail*, ou seja qualquer pessoa pode executar seu próprio servidor XMPP e não há nenhum servidor central. A Internet Engineering Task Force formalizou o XMPP como um protocolo de mensagens instantâneas e tecnologia de presença. Através das normas XEP novas funcionalidades personalizadas podem ser construídas em cima do XMPP, tornando-o um protocolo flexível.

O facto de ser um protocolo específico para IM e de ser bastante flexível faz com que seja a opção mais acertada para a realização do projecto.

3.2 Servidores XMPP

Nesta secção são abordados os servidores XMPP mais relevantes e as suas características, sendo efectuada uma análise crítica com o intuito de avaliá-los como base para o desenvolvimento do projecto.

3.2.1 Introdução

O objectivo deste trabalho passa pela implementação de um serviço de presença na Internet que possibilite aos utilizadores a comunicação, em tempo real, entre si e com outras plataformas, para além daquela em que estão incluídos. Para permitir esta comunicação, vai ser usado o modelo cliente-servidor com o protocolo XMPP como padrão de transferência de dados.

Existem várias soluções na área de servidores de *Instant Messaging* com suporte para o protocolo XMPP. Um estudo [Dan10] foi realizado sobre as soluções mais relevantes tendo em conta alguns parâmetros como a escalabilidade ou a abrangência de implementação.

A abrangência e qualidade de implementação pode ser medida com base em dois factores, as extensões XEP e os RFC. O RFC é um *memorandum* publicado pela IETF descrevendo métodos de trabalho na Internet [AMS10]. Alguns destes RFC são adoptados como padrões. Existem alguns RFC que são a base das especificações do Extensible Messaging and Presence Protocol (XMPP) e que formalizam o núcleo de protocolos desenvolvidos pela *Jabber open-source community* em 1999 [Fou10b].

A XMPP Standards Foundation (XSF) desenvolve extensões para o XMPP através de um processo normalizado centrado em XMPP Extension Protocols (XEPs) [Fou10a]. Sabendo que uma aplicação está em conformidade com um determinado XEP, infere-se que esta obedece a uma norma aprovada e que tem a funcionalidade descrita nesse XEP.

3.2.2 Servidores e Características Globais

Grande parte dos servidores que implementam o protocolo XMPP destinam-se à comunicação instantânea, também conhecida como IM. Embora o XMPP seja um protocolo desenvolvido com os *Instant Messengers* no horizonte as suas características podem ser usadas para outras finalidades. De um estudo [Dan10] prévio existente das características de cada servidor destacaram-se 3 servidores. As características mais relevantes para a escolha passam pela abrangência de implementação (quantidade de funcionalidades implementadas), a escalabilidade (a aplicação final necessita de comportar um grande número de utilizadores) e facilidade de extensibilidade (capacidade de se implementar rapidamente funcionalidades além das já implementadas).

O Ejabberd[1] é uma das 3 hipóteses que se destacaram. Em termos de extensibilidade esta ferramenta é código aberto e possui uma arquitectura modular, o que torna mais fácil a inclusão de novas funcionalidades. O facto de ser modular também permite que durante a instalação sejam só adicionados os módulos pretendidos. O Ejabberd escala naturalmente para atender milhares de utilizadores em simultâneo num único nó Ejabberd. Adicionando nós em *clusters*, as organizações podem não só multiplicar o número dos seus membros como também aumentar a tolerância a falhas do sistema. Se um *cluster* falhar, o desempenho da aplicação de *instant messaging* não é afectada. Igualmente, *clusters* individuais podem ser adicionados ou removidos instantaneamente sem parar os servidores [Pro10a]. Um exemplo de uma implementação desta aplicação é o servidor XMPP jabber.org que tem uma base de dados de 330 000 utilizadores e aproximadamente 15 000 utilizadores conectados em qualquer instante [Sai09a]. Devido às suas características, esta é uma tecnologia válida para o projecto que se pretende implementar. Poderá mesmo dizer-se que é a tecnologia que, à partida, é mais popular e completa.

O Openfire[2] é outro dos servidores que se destacaram. De notar que, apesar de ser uma ferramenta de código aberto, para usar o *plugin* que possibilita *clustering* é necessário adquirir uma

[1] http://www.ejabberd.im/

[2] http://www.igniterealtime.org/projects/openfire/

licença Oracle Coherence. O *clustering plugin* adiciona suporte para a execução de vários servidores Openfire redundantes num *cluster*. Executando Openfire num *cluster* distribui-se a carga entre vários servidores e obtém-se redundância para o caso de um dos servidores falhar [Dom09]. Um estudo indica que a versão 3.2 do Openfire suporta mais de 50 000 utilizadores numa máquina modesta [Sof10]. A versão mais recente da aplicação é a 3.6. Esta aplicação, desenvolvida pela Jive Software, possui uma comunidade bastante alargada, encontra-se bem documentada e tem uma arquitectura modular, o que torna o alargamento das suas funcionalidades bastante célere. Esta é outra implementação bastante popular dentro das estudadas. É uma aplicação com inúmeras funcionalidades e o facto de manter a escalabilidade desejada faz com que cumpra os requisitos necessários para ser considerada como uma opção válida.

Finalmente o Tigase Server[3] classifica-se como um Servidor XMPP leve e escalável [Hef07]. O servidor Tigase permite um elevado tráfego de informação e um grande número de sessões de utilizador numa única máquina. Um dos números mencionados num estudo feito foi o de 500 000 utilizadores simultâneos numa única máquina [Tig09b]; o outro foi de 1 milhão de utilizadores ligados em *cluster* [Tig09a]. Os números relativos à escalabilidade, o facto de haver uma comunidade activa e de ser uma tecnologia madura fazem com que esta seja uma aplicação que preenche todos os requisitos para ser um bom ponto de partida de implementação.

3.2.3 Conclusão

Foram investigadas diversas implementações, umas mais completas que outras, umas ainda numa fase inicial de desenvolvimento e outras mais maduras, e verificou-se que existem várias possibilidades válidas para servir como ponto de partida para o protótipo a desenvolver numa fase posterior deste trabalho.

3.3 Clientes *Web* XMPP

Seguindo a exploração de tecnologias que podem ser usadas no projecto, foram investigadas as funcionalidades de alguns clientes *web* que suportam o protocolo XMPP [Dan10]. As características exploradas vão desde funcionalidades, a questões de usabilidade consideradas importantes assim como à facilidade de alteração. O cliente destina-se a ser integrável com a interface do serviço de correio electrónico, devendo este facto também será tomado em conta.

As duas aplicações mais promissoras são a iJabBar e a JWChat. A iJabBar possui uma interface compacta, o que permite uma integração mais fácil com a página *Web* e funciona em vários *browsers*. A iJabBar não tem implementada a gestão de contas de outros protocolos nem a gestão da lista de contactos. A iJabBar é uma aplicaçao gratuita e *open-source* desenvolvida em GWT, ou seja, é implementada em Java, e posteriormente, convertida para JavaScript.

A JWChat tem uma licença GPL2. É um *software* bastante maduro e estável, mas a sua comunidade e utilização tem vindo a decrescer. Para além das funcionalidades básicas de IM,

[3] http://www.tigase.org/

este *software* possibilita criar salas de conversação de múltiplos utilizadores, armazenamento das preferências do lado do servidor e armazenamento do histórico de mensagens (necessita de apoio do servidor). Tem suporte para transportes para outros serviços de IM como ICQ, AIM e MSN. O maior defeito deste cliente é o facto de sua interface não ser integrável com outras páginas *Web*.

Concluindo, é possível verificar que na área de clientes *web* XMPP existe uma grande variedade de aplicações implementadas e também bibliotecas que permitem a sua implementação. A solução iJabBar será, à partida, aquela que mais se adequa a ser incluída no projecto, sobretudo devido à facilidade de integração. Qualquer que seja a escolha, esta terá de ser adaptada de maneira a permitir adicionar as funcionalidades desejadas.

3.4 Exemplos de APIs de Plataformas Integráveis no Projecto

API é o acrónimo de *Application Programming Interface* ou, em Português, Interface de Programação de Aplicações.

É a interface, normalmente documentada, que uma biblioteca ou *framework* disponibiliza para que o programador possa utilizá-la. Esta interface é o conjunto de padrões de programação que permite a construção de aplicações e a sua utilização independentemente da sua implementação. Uma API funciona através da comunicação entre diversos sistemas, definindo assim comportamentos específicos de determinados objectos numa interface. Ou seja, a API irá interligar diversas funções de um *site* de modo a possibilitar que possam ser utilizadas noutras aplicações.

Como foi descrito na Introdução, o objectivo deste trabalho será juntar a informação de várias fontes e apresentá-la ao utilizador na mesma interface, sendo possível a este interagir com estas fontes numa mesma aplicação. Para atingir este objectivo é preciso recolher a informação das várias fontes através das API que estas disponibilizam. As API com interesse nesta dissertação são de plataformas que permitem algum tipo de interacção e comunicação entre os seus utilizadores.

Uma característica importante, para esta dissertação, das API são os limites de pedidos. Uma vez que o funcionamento das API se baseia no método de *pull*, é necessário efectuar pedidos com um certo período, de maneira a verificar a ocorrência de nova informação. Os limites das interfaces das plataformas Twitter, Facebook e Last.fm permitem uma boa aproximação a tempo real. A API do Last.fm tem um limite de uso de 1 pedido por segundo, a do Twitter impõe um máximo de 150 pedidos à API por hora (2.5/min) e o Facebook não impõe limites.

Estas API permitem a recolha e a submissão de nova informação. Exemplos do uso e mais informação acerca destas interfaces de aplicações podem ser visualizadas em [Dan10].

3.5 Conclusão

Do estudo efectuado, conclui-se que o protocolo XMPP e as tecnologias existentes permitem a implementação com sucesso e qualidade do sistema proposto. Alguns dos servidores e clientes

estudados podem ser um bom ponto de partida dadas as suas qualidades. A recolha de informação através das API também é possível tendo em conta a escalabilidade pretendida e também as funcionalidades desejadas.

Capítulo 4

Evolução para Comunicação em Tempo Real de Informação Dispersa

Neste capítulo apresenta-se a proposta de solução elaborada para abordar os objectivos definidos. Inicialmente são enumerados os requisitos e é apresentada a arquitectura da solução. São abordados todos os processos definidos para o funcionamento da solução. Finalmente, são apresentadas as conclusões retiradas desta implementação.

4.1 Introdução

É importante ter em mente o contexto desta solução: é uma abordagem a ser desenvolvida para depois ser aplicada e integrada com o sistema da Portugalmail. Além de ser definido o modo de funcionamento também são tidas em conta as tecnologias existentes que podem ser usadas e adaptadas para possibilitar a evolução do correio electrónico para modelos de comunicação em tempo real.

4.2 Requisitos

A abordagem apresentada nesta dissertação está intrinsecamente ligada a um produto. Nesta secção são apresentados os requisitos funcionais e não funcionais que foram tidos em conta na elaboração do trabalho e que, finalmente, levaram à elaboração de um protótipo.

O requisito principal pode dividir-se em duas partes: a primeira é a visualização instantânea de nova informação por parte do utilizador, sendo que esta informação pode ser proveniente de inúmeras fontes, incluindo o correio electrónico; a segunda é a interacção com as fontes tendo como ponto de partida a informação recebida. Exemplificando, esta interacção pode passar por uma simples resposta a uma mensagem recebida ou pela marcação de uma mensagem de correio electrónico como lida. Estas acções variam dependendo das fontes e das possibilidades que estas

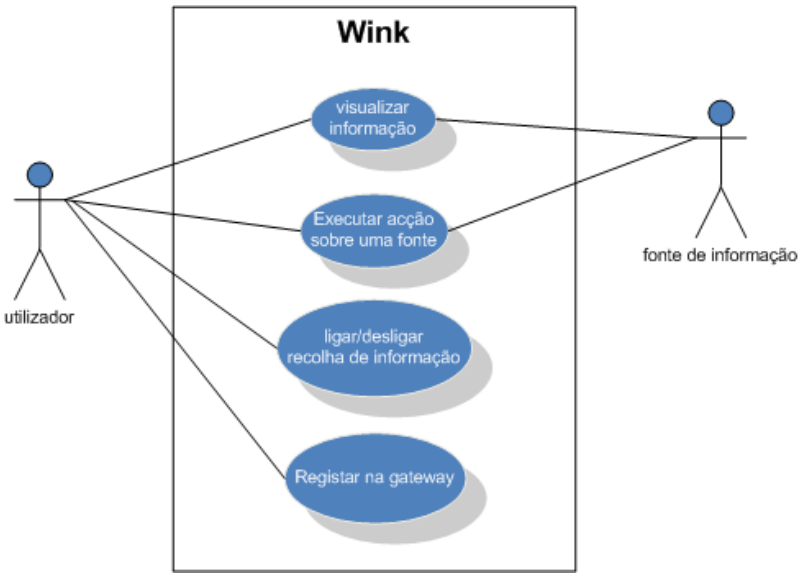


Figura 4.1: Casos de uso principal.

disponibilizam. O sistema terá que recolher os dados necessários para aceder às fontes, por exemplo senhas, da base de dados da plataforma de correio electrónico que se encontra na rede local. O utilizador terá a possibilidade de activar ou desactivar a recolha de informação de determinada fonte. Estes requisitos encontram-se ilustrados no caso de uso presente na Figura 4.1.

Com base no requisito principal, foi proposto usar a metodologia cliente-servidor e o protocolo de comunicação ser o XMPP. O núcleo do sistema será um servidor ou *cluster* de servidores que irá recolher a informação e interagir com as fontes. A interface será um cliente que comunicará com o núcleo. Este processo é apresentado na Figura 4.2.

A proposta de solução deverá comunicar com outros sistemas, incluindo aqueles que usem outros protocolos de comunicação. Entre estes protocolos dá-se ênfase a protocolos de IM, como por exemplo o MSN.

O sistema terá que integrar com as plataformas da Portugalmail de diferentes formas. Os utilizadores do sistema serão os mesmos das plataformas da empresa e as listas de contactos destes serão elaboradas com base no gestor de contactos que a empresa fornece aos utilizadores. Duas fontes possíveis para o sistema serão a plataforma de correio electrónico e a plataforma de *Blog* da empresa. O servidor deverá correr em Linux e a interface deverá ser uma aplicação *web* para integrar com as plataformas da empresa.

Além destes requisitos funcionais destaca-se a importância de alguns requisitos não funcionais. O sistema deverá ser seguro, de tal modo que um utilizador só deverá ter acesso a dados que a si digam respeito, incluindo informação recolhida das várias fontes. A solução deverá ter um desempenho aceitável para 50 000 utilizadores e estes deverão obter respostas em tempos aceitáveis. Finalmente, o último requisito não funcional a ser destacado é a usabilidade: a apresentação dos dados deverá ser escalável de maneira a que o utilizador consiga a visualização de toda a infor-

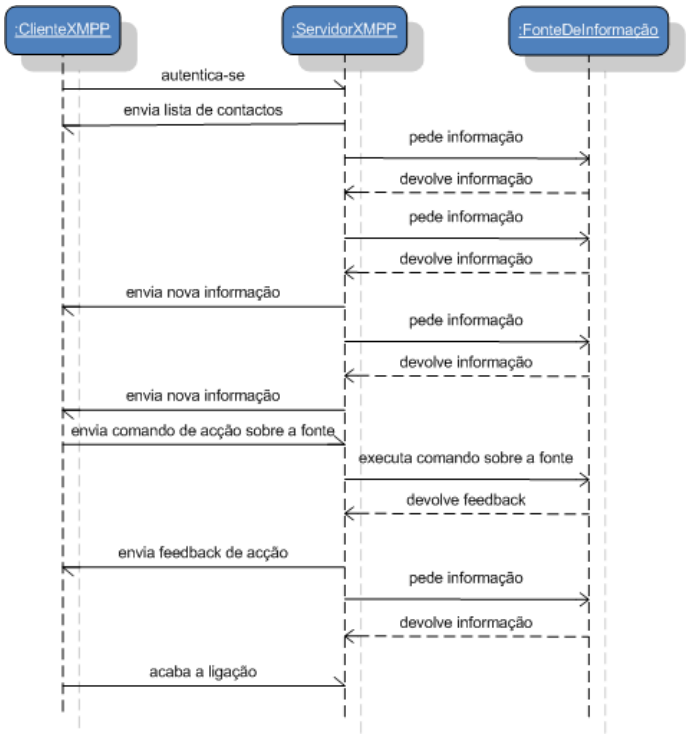


Figura 4.2: Diagrama de sequência para o cenário de utilização principal.

mação de uma forma usável e agradável. O utilizador não deverá ter a necessidade de se deslocar para outras interfaces ou plataformas para visualizar os conteúdos das mensagens.

4.3 Arquitectura Proposta

Esta Secção apresenta a arquitectura da abordagem proposta, todas as componentes da abordagem e a forma como comunicam entre si. São apresentados os elementos da solução proposta, inclusive, os dois que têm que sofrer alterações e quais as características destas.

4.3.1 Solução

A solução é um sistema com vários componentes ligados em rede que comunica com outros elementos numa rede local e com sistemas espalhados pela Internet. A interface do sistema, também referido como cliente XMPP, não se encontra à partida na rede local. Na Figura 4.3 encontra-se o diagrama de arquitectura do sistema. As comunicações entre os componentes serão efectuadas por *sockets* TCP. De salientar que o protocolo de comunicação entre o cliente e o servidor XMPP será o Bidirectional-streams Over Synchronous HTTP (BOSH) que possibilita a troca de *stanzas* sobre o HTTP como está definido no XEP-0206 (XMPP Over BOSH). O BOSH

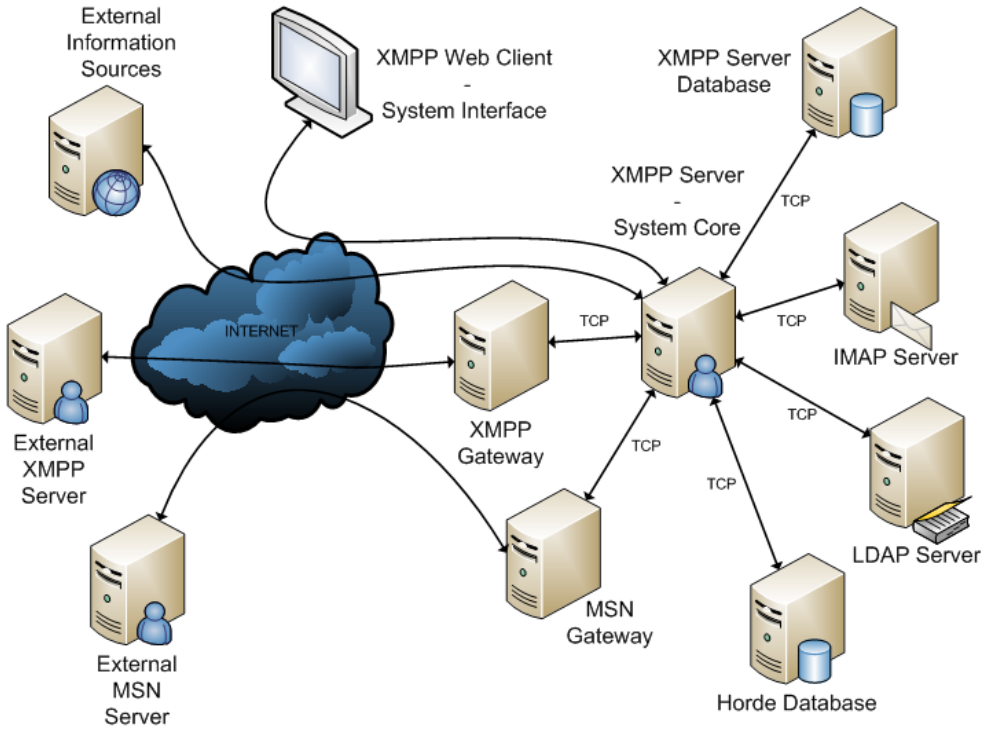


Figura 4.3: Arquitectura global do sistema.

simula uma ligação *socket* TCP de longa duração sobre o HTTP através de uma sequência de troca de mensagens [PSSM09]. A função de cada componente é explicada nas secções seguintes.

4.3.2 Servidor XMPP

Uma vez que se irá usar como base um servidor XMPP e dado o facto dos servidores existentes serem modulares, nesta secção apenas será apresentada uma arquitectura do servidor XMPP com os componentes necessários para a comunicação com os módulos a serem implementados, sendo estes descritos integralmente. Para cada fonte terá de se implementar um módulo, seguindo a mesma estrutura, onde os componentes que comunicam com a fonte alteram os seus métodos mas não o seu funcionamento.

Como se pode ver na Figura 4.4 os módulos funcionam em conjunto com o servidor XMPP e a base de dados do Horde. O servidor XMPP por sua vez também possui uma base de dados própria. O servidor tem 3 componentes que interagem com os módulos: o ConnectionManager, que é o responsável por controlar as ligações dos utilizadores ao servidor, o MessageRouter, que é o responsável por encaminhar as mensagens recebidas para os devidos destinatários, e o DatabaseManager, que controla as ligações do servidor efectuadas à base de dados. Os módulos são divididos em 4 componentes: MessageParser, ActionsManager, CommandManager e DataRetriever.

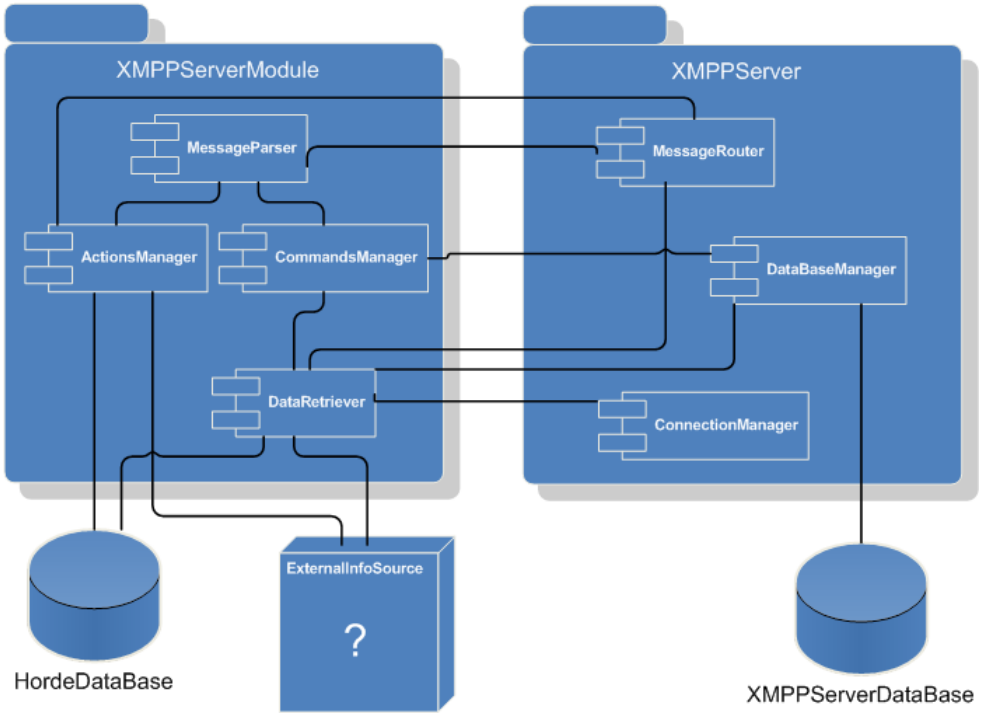


Figura 4.4: Arquitectura dos módulos do servidor XMPP.

O componente DataRetriever recebe informação de conexão enviada pelo servidor. Quando um utilizador se conecta ou desconecta do sistema este componente activa ou desactiva a recuperação de informação. Se o seu estado for activo, este componente irá proceder à recuperação de informação da fonte. Na eventualidade dessa informação ser mais recente que o momento em que o componente se activou ou mais recente que a última actualização então será reencaminhada para o MessageRouter do servidor no formato apropriado. Para recuperar a informação do utilizador que se encontra conectado, este componente poderá necessitar de dados previamente introduzidos pelo utilizador na base de dados do Horde. Nessa situação, o componente irá efectuar a ligação à base de dados do Horde e retirar os dados necessários. O funcionamento do DataRetriever também é controlado pelas definições guardadas pelo utilizador na base de dados do servidor. No entanto, se o utilizador tiver nas definições que a recolha está desactivada então esta não se irá efectuar.

O MessageParser é o componente que analisa todas as mensagens que são dirigidas aos módulos e reencaminha os dados extraídos para o componente respectivo, dependendo das mensagens indicar uma acção sobre a fonte ou um comando para o módulo.

O CommandManager é o componente que realiza acções sobre o módulo. A acção que está disponível em todos os módulos é a acção de activar ou desactivar a recuperação de informação. Dependendo da fonte, poderão ser definidos outros comportamentos para o módulo. Além das tabelas que permitem o funcionamento normal do servidor XMPP, é adicionada uma tabela à base de dados do servidor para guardar as definições de cada utilizador para cada módulo. Este

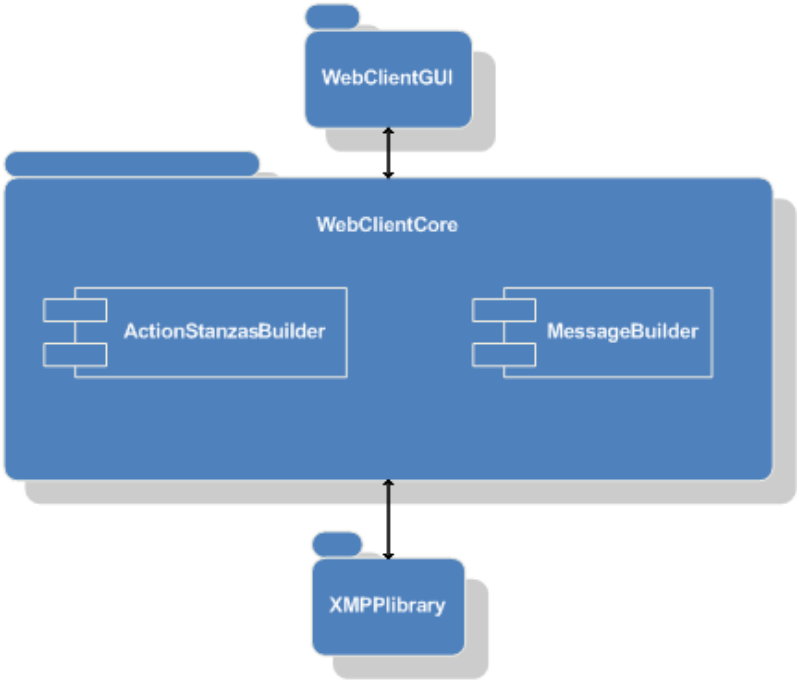


Figura 4.5: Arquitectura do cliente XMPP.

componente utiliza o DatabaseManager do servidor para guardar estas definições e permitir a sua persistência.

Finalmente o ActionsManager é o responsável por efectuar acções sobre a fonte. Para efectuar essas acções, o componente pode necessitar de dados existentes na base de dados do Horde e, como tal, existe uma ligação a esta. O ActionsManager pode recuperar informação da fonte tal como o DataRetriever; a diferença entre os dois é que no DataRetriever a recuperação é automática e periódica enquanto que no ActionsManager é unicamente a pedido do utilizador. Além de recuperar informação, também tem a capacidade de enviar informação para a fonte como um comentário ou actualizar o seu estado.

4.3.3 Cliente XMPP

Outro dos elementos que necessitam de ser modificados para implementar a proposta de solução é o cliente *Web* XMPP. As alterações a efectuar no cliente não são estruturais e a arquitectura do cliente manter-se-á inalterada. Apenas serão adicionadas dois componentes ao cliente como se pode ver na Figura 4.5. O objectivo da classe MessageBuilder será receber as *stanzas* do tipo *message* provenientes do servidor XMPP e editar o seu *body* de maneira a adicionar à mensagem as opções de acção e tornar o seu conteúdo auto-contido. Exemplificando, se o cliente receber a *stanza*:

```
1 <message from='romeo@montague.lit' to='juliet@capulet.com' type='chat' source='
    VeronaArquives'>
2   <body>
```

```
3       &lt;b&gt;Our photo!&lt;/b&gt;
4       &lt;br/&gt;
5        http://etc.usf.edu/clipart/5700/5767/romeo&amp;juliet_6_lg.gif
6     </body>
7     <role id='photomessage'/>
8     <shelf id='rj101'/>
9   </message>
```

A classe MessageBuilder, sabendo que a fonte da mensagem é *VeronaArquives* e que o tipo de mensagem é *photomessage*, transforma a mensagem no seguinte código HTML para enviar para a interface[1]:

```
1  <div class="VeronaArquivesStyle">
2    <b>Our photo!</b>
3    <br/>
4    <img alt="Romeo and Juliet" src="http://etc.usf.edu/clipart/5700/5767/romeo&
         juliet_6_lg.gif" />
5    <br/>
6    <form name="formulverona1">
7          <textarea name=verona1 cols=40 rows=3 />
8    </form>
9    <a href="javascript:veronaArquives.sendShelfComment('juliet@capulet.com','
         romeo@montague.lit','rj101', document.formulverona1.verona1.value)"
         class="VeronaArquivesStylebtn">Comment Shelf!</a>
10 </div>
```

A classe ActionStanzasBuilder vai fornecer as funções que são chamadas para efectuar acções sobre as fontes. Ou seja, esta classe vai construir as *stanzas* que são enviadas para os módulos do servidor. Após esta construção, as *stanzas* são enviadas para o servidor com recurso à biblioteca XMPP. Exemplificando, ao chamar a função:

```
1  veronaArquives.sendShelfComment('juliet@capulet.com','romeo@montague.lit','
       rj101','we look great :)')
```

Que estará implementada na classe ActionStanzasBuilder, será construída a seguinte *stanza*:

```
1    <iq xmlns="jabber:client" from="juliet@capulet.com" to="romeo@montague.lit"
         type="set" id="abc" >
2      <query xmlns="VeronaArquives:photo:addcomment"/>
3      <shelf id="rj101"/>
4      <comment>we look great :)</comment>
5    </iq>
```

As especificações da construção das *stanzas* XMPP usadas nestes exemplos encontra-se nas Secções 4.5 e 4.6.

[1] Diferentes tipos de mensagem vão ter diferentes opções.

4.4 Exploração das Funcionalidades e Tecnologias do XMPP

O XMPP é um protocolo de mensagens instantâneas, com variadíssimas extensões, sobre o qual serão definidas as novas funcionalidades especificadas na proposta de solução. Além do funcionamento *core* vão ser usadas as extensões necessárias ao funcionamento habitual de um servidor e cliente *Web*, como é o caso, por exemplo, da extensão de gestão de contactos ou ligação a serviços externos. Não irão ser abordadas todas as especificações do XMPP nem as suas extensões neste relatório; embora estas sejam importantes, apenas serão abordadas alterações ou extensões ao XMPP para permitir o conceito que se deseja provar.

O servidor XMPP, o cliente XMPP e a *gateway* são tecnologias que implementam algumas especificações do XMPP e que fazem parte da arquitectura. Serão necessárias especificar alterações ao funcionamento normal do servidor e do cliente mas o funcionamento das *gateways* irá manter-se inalterado.

Falando genericamente das funcionalidades aproveitadas do servidor e cliente XMPP, estas permitirão a comunicação em tempo real entre dois utilizadores, darão a possibilidade do utilizador saber o estado de outro utilizador e da sua lista de contactos assim como permitirão saber se este se encontra *online*. Durante as próximas secções serão abordadas outras funcionalidades já existentes e outras que serão adicionadas.

A *gateway* serve de intermediário entre protocolos diferentes. De um lado encontra-se o sistema XMPP e do outro um protocolo de IM. O objectivo da *gateway* é usar uma conta de outro sistema que fale XMPP ou outro protocolo qualquer de IM de maneira transparente para o utilizador. Ou seja, são adicionados à lista de contactos do utilizador do sistema XMPP todos os contactos da conta do outro sistema sem proceder a qualquer alteração na sua lista de contactos, como se pode ver na Figura 4.6. De um ponto de vista simplista, do lado do servidor XMPP, a *gateway* funciona como outro servidor XMPP ao qual é possível efectuar a ligação para falar com os seus utilizadores; do lado do servidor de um outro protocolo, a *gateway* funciona como um cliente único que fala esse outro protocolo e que comunica com outros utilizadores desse protocolo. Além deste funcionamento simplista a *gateway* oferece ao servidor XMPP a capacidade de se registar, fornecendo as credenciais de autenticação da conta do outro protocolo, e a gestão da lista de contactos da conta. O uso de várias *gateways* irá permitir a ligação com inúmeras contas que o utilizador tenha, ajudando assim à agregação de informação.

4.5 Agregação da Informação de Várias Fontes

O elemento do sistema encarregue de agregar a informação é o servidor XMPP e o elemento encarregue de apresentar a informação ao utilizador é o cliente *Web*. Por conseguinte, pode-se dividir este processo em dois passos. O primeiro será a recolha dos dados efectuada pelo servidor às várias fontes e o segundo o envio da informação para o cliente *Web* e as consequências da forma de envio na usabilidade e integração de informação.

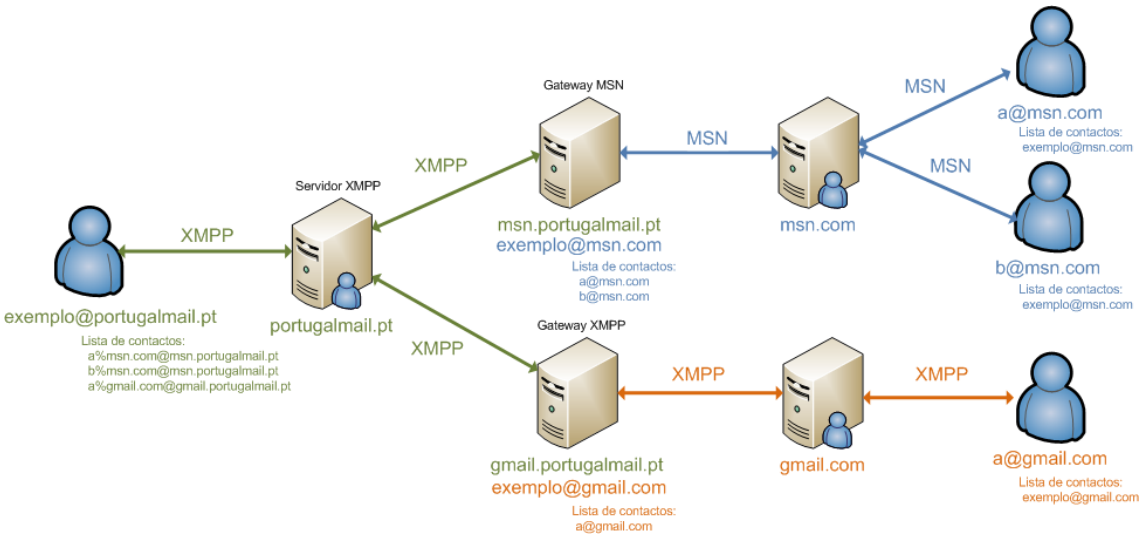


Figura 4.6: *Gateways* e listas de contactos.

As fontes da informação podem ser diversas, desde API, bases de dados, servidores de correio electrónico (IMAP, POP3, *etc.*); entre outras. O *cluster* de servidores fica responsável pela recolha da informação para os seus utilizadores. Existem dois métodos de recolha de informação: *push*, onde as fontes enviam informação nova para o servidor XMPP quando a têm, e *pull*, onde o servidor XMPP requisita a informação às fontes sendo que esta informação poderá ser nova, antiga ou uma mistura. Em termos de carga sobre o servidor XMPP é preferível o método *push* uma vez não há a necessidade deste verificar continuamente se existe informação nova. Mas, uma vez que as fontes são maioritariamente independentes do sistema proposto, o método escolhido terá de ser o que a fonte permite. Se a recolha for por *push* então o servidor terá de se registar na fonte e quando receber informação verificar para que utilizador se destina essa informação. Se a recolha for por *pull* o servidor terá de pedir a informação com um período pré-definido.

Depois de recolher a informação, esta será enviada para o cliente *Web* no formato XMPP. Usando a extensão XEP-0166 é possível efectuar a transferência de diferentes tipos de dados como som, vídeo ou qualquer tipo de ficheiro [LBS+09], mas a maioria da informação enviada será no formato de texto. De maneira a enviar texto estruturado e formatado, este será convertido para entidades HTML e reconvertido antes de ser apresentado ao utilizador; através do HTML será facilmente formatada a apresentação dos dados uma vez que estes serão visualizados num *browser*. Para enviar informação para o cliente *Web* são usadas as *stanzas* de mensagem, referidas na secção 3.1.3. O Servidor cria uma *stanza* de mensagem e envia-a para o cliente definindo os respectivos atributos. O formato destas *stanzas* é o seguinte:

```
1 <message from='romeo@montague.lit' to='juliet@capulet.com' type='chat' source='
    VeronaArquives'>
2   <body>
3     &lt;b&gt;Our photo!&lt;/b&gt;
4     &lt;br/&gt;
```

```
5       &lt;img  alt=&quot;Romeo and Juliet&quot; src=&quot;http://etc.usf.edu/
            clipart/5700/5767/romeo&amp;juliet_6_lg.gif&quot;&gt;
6     </body>
7     <shelf id='rj101'/>
8   </message>
```

Dentro do atributo *body* o servidor coloca a mensagem devidamente codificada, no atributo *to* coloca a JID do receptor, no atributo *from* o servidor coloca a JID emissor da mensagem e no atributo *source* é colocada a fonte da mensagem.

O atributo *source* é opcional. O valor deste indica a origem da mensagem (de que fonte é que ela provém) para o cliente poder definir a forma de interacção, tal como será descrito na secção seguinte. Se não estiver definido este atributo então o cliente identificará a mensagem como proveniente de um contacto de IM.

Além da *tag body*, definida pelo XMPP, será possível adicionar mais *tags* filhas à *stanza* do tipo *message*. Estas *tags* contêm informação necessária para identificar a informação e para possibilitar a resposta.

O emissor que é colocado irá variar dependendo da fonte e das configurações do receptor, sendo este processo de escolha efectuado pelo servidor. O motivo da variação do emissor está relacionado com razões de usabilidade e agregação de informação, ou seja, o utilizador tem a possibilidade de agrupar contactos. Por exemplo, a um contacto de IM pode agrupar um amigo da sua conta do Twitter e, na mesma janela de *chat* do cliente *Web*, recebe as mensagens instantâneas do contacto assim como as actualizações do Twitter que entretanto o contacto realize. O cliente dispõe de uma interface onde poderá agrupar contactos das várias contas que tenha adicionado ao sistema (o método de adição de contas é explicado na secção 4.7). Devido a questões de escalabilidade, para não serem criadas janelas de *chat* para cada um dos contactos ou conexões das fontes, é criado um contacto especial na lista de contactos do cliente *Web* para cada uma das fontes. Todas as novas actualizações dos contactos ou conexões dessa fonte são enviadas para essa janela de *chat*. Com base nos agrupamentos efectuados, o servidor define no atributo *from* se a mensagem é enviada em nome da fonte ou em nome de um agrupamento que simboliza um contacto único. Exemplificando, a Juliet tem na sua lista de contactos *romeo@montague.lit* e agrupa a este contacto a conta do Romeo que está nos seus amigos do Twitter. Quando o Romeo actualizar o seu estado do Twitter o atributo *from* dessa nova mensagem enviada à Juliet será definido com o valor *romeo@montague.lit*. Se a Juliet não tivesse agrupado as contas então o atributo *from* seria definido com o valor *twitter@twitter.com* que é o contacto criado para fonte.

Concluindo, servidor recolhe a informação e envia-a para o cliente na forma de uma *stanza* <message/>. O emissor da *stanza* é definido pelo servidor com base nos agrupamentos do utilizador.

4.6 Processamento dos Comentários Efectuados à Informação

Enquanto na secção anterior se abordou a chegada de informação, nesta será abordado o envio de mensagens. Como referido, serão apresentadas ao utilizador dois tipos de janelas diferentes de *chat*: a janela de uma fonte e a janela de um contacto ou grupo de contactos. De ambas as janelas é possível enviar dois tipos de *stanzas*, tipo *message* e tipo *iq*, mas dependendo da janela estas terão repercussões diferentes. Mantendo as funcionalidades do cliente *Web*, as mensagens de cada janela são enviadas para o servidor com o formato de *stanza* do tipo *message*. Estendendo as funcionalidades do cliente, adicionando botões de acções à informação recebida, enviam-se *stanzas* do tipo *iq* para o servidor XMPP.

As *stanzas* do tipo *message* recebidas pelo Servidor XMPP em que o receptor definido é outro utilizador (enviadas de uma janela de contacto), são reencaminhadas para esse utilizador. As que têm o receptor definido como uma fonte são analisadas pelo servidor; o seu propósito é emitir comandos ao servidor. Esses comandos servem para activar ou desactivar a recolha de informação, executar operações sobre a fonte ou outra acção para a qual não sejam necessários dados além de uma possível mensagem introduzida pelo utilizador. Para implementar este comportamento apenas se altera o servidor, a estrutura das *stanzas* mantém-se inalterada.

As *stanzas* do tipo *iq* enviadas através de uma janela têm sempre como receptor uma fonte. São usadas, tal como as *stanzas* do tipo *message* destinadas a fontes, para o servidor executar acções sobre a fonte. A diferença é que o servidor envia uma resposta informando do sucesso da acção e permitem o envio de dados na forma de atributos.

Exemplo de *iq* enviada do cliente para o servidor XMPP (a JID de destinatário é o serviço veronaarquives do servidor capulet.com):

```
1  <iq xmlns="jabber:client" from="juliet@capulet.com" to="romeo@veronaarquives.
       capulet.com"  type="set" id="abc" >
2    <query xmlns="VeronaArquives:photo:arquive"/>
3    <shelf id="rj101"/>
4    <packetto>romeo@montague.lit</packetto>
5  </iq>
```

Exemplo da *iq* de resposta enviada pelo servidor no caso de sucesso:

```
1  <iq from="romeo@veronaarquives.capulet.com" to="juliet@capulet.com" id='abc'
       type='result'/>
```

Exemplo da *iq* de resposta enviada pelo servidor no caso de erro:

```
1  <iq from="romeo@veronaarquives.capulet.com" to="juliet@capulet.com" id='abc'
       type='error'>
2    <error type='cancel'>
3      <service-unavailable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
4    </error
5  </iq>
```

Para concluir o processo, o servidor fica encarregue de efectuar a acção na fonte, o que dependerá do tipo de fontes que o sistema integrará.

Por motivos de usabilidade, a construção das *stanzas* ficará a cargo do cliente, cabendo ao utilizador preencher apenas a mensagem a enviar à fonte.

4.7 Integração de Dados com Plataformas da Empresa

Como referido na introdução, existe a necessidade de integração com as plataformas da Portugalmail. Esta integração de dados é efectuada em quatro níveis: na criação dos utilizadores, nos dados necessários para aceder às fontes de cada utilizador, nas relações entre utilizadores e na transferência de informação. As ligações às bases de dados da rede interna são realizadas directamente, sem recurso a serviços *Web*, para tornar este processo mais rápido.

Os utilizadores da solução são importados da base de utilizadores da empresa. Estes dados são disponibilizados por um servidor de Lightweight Directory Access Protocol (LDAP) que, fundamentalmente, é uma base de dados com características que permitem um bom desempenho quando se efectuam poucas alterações e muitas pesquisas. O servidor XMPP vai usar o servidor LDAP como fonte de autenticação. Assim, quando for criado um utilizador no sistema da empresa, ele fica imediatamente identificado como utilizador a solução. Resumidamente, a gestão de utilizadores é completamente autónoma da solução uma vez que esta só necessita de aceder à informação dos utilizadores, obtendo-se deste modo sincronização entre os utilizadores da solução e os do sistema da empresa.

Toda a informação respeitante a outras contas dos utilizadores é inserida na plataforma de gestão de contactos da empresa como, por exemplo, as credenciais de autenticação ou qualquer outra informação necessária para recuperar informação das fontes, é guardada na sua base de dados. O servidor XMPP, quando necessite desta informação para interagir com as fontes, acede directamente à base de dados uma vez que esta se encontra na rede local.

Cada utilizador tem uma lista de contactos que é preenchida pelo servidor XMPP. Os elementos desta lista podem ser contactos de outras contas de IM, como é abordado na secção 4.4, ou contactos definidos pela plataforma de gestão de contactos da empresa. A plataforma de gestão de contactos permite ao utilizador criar uma rede social de contactos através do envio de pedidos de ligação a outros utilizadores. Esta informação de conexões encontra-se na base de dados do Horde. Para obter os contactos de um utilizador o servidor XMPP liga-se a esta base de dados e extrai a informação de duas tabelas com a estrutura que se pode ver na Figura 4.7. Para ser considerado um contacto é necessário haver uma ligação bilateral entre os utilizadores: o utilizador A tem que aceitar o pedido de B e o utilizador B tem que aceitar o pedido de A.

Finalmente, existirá uma ligação com componentes do sistema da empresa para trocar informação, ou seja, funcionarem como fonte. Um exemplo será o servidor de IMAP onde estão armazenadas as mensagens de correio electrónico dos utilizadores. Os identificadores de utilizador da solução são os mesmos que os identificadores da conta de *mail*, logo estas duas contas estão automaticamente associadas. Procede-se à verificação directa periódica de mensagens novas e, caso existam, são apresentadas na janela de utilizador no cliente *Web*. Sobre esta mensagem será possível efectuar acções como, por exemplo, marcá-las como lidas. Esta informação é enviada do

	contact_id [PK] integer	contact_idowner character varying(255)
1	1	ruicarneiro@portugalmail.pt
2	2	goncalo@portugalmail.pt
3	3	hector@portugalmail.pt
4	4	dansi@portugalmail.pt

(a) Tabela sybil_contact

	connection_requester_id integer	connection_receiver_id integer	connection_status connection_status
1	2	1	accepted
2	1	3	accepted
3	3	1	pending
4	1	4	accepted
5	4	1	accepted

(b) Tabela sybil_connection

Figura 4.7: Exemplo de tabelas da plataforma de gestão de contactos da Portugalmail.

cliente *Web* para o servidor XMPP através da Internet e depois do servidor XMPP para o servidor IMAP através da rede interna.

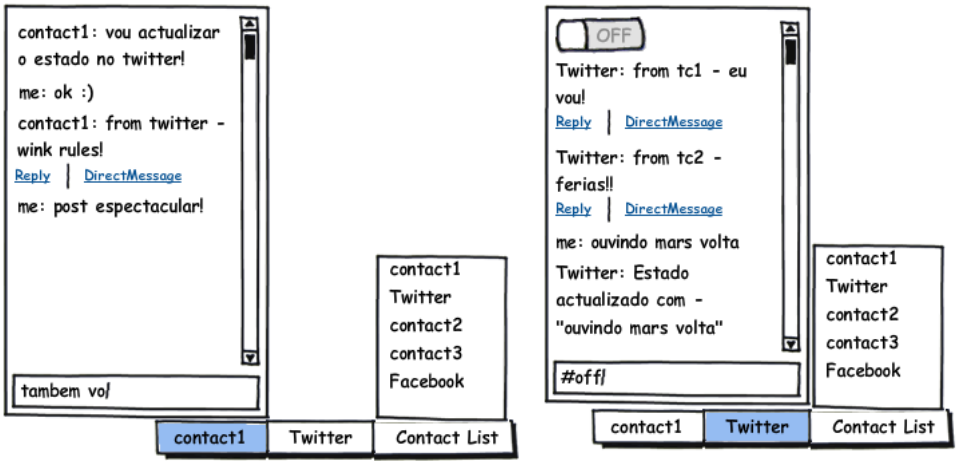
Concluindo, qualquer troca de informação entre bases de dados ou servidores situados na rede interna é efectuada acedendo directamente a esses serviços através do seu protocolo. As ligações aos servidores de IMAP, LDAP e base de dados do Horde são necessárias mas, além dessas, podem ser estabelecidas muitas mais dependendo das funcionalidades desejadas no futuro.

4.8 Interface da Solução e Integração com as Plataformas da Empresa

Uma vez que as aplicações da empresa apresentam interfaces *Web* e a interface da solução necessita de se integrar com elas, torna-se necessário que esta seja implementada com tecnologia *Web*. Para obter as funcionalidades pretendidas, uma vez que a solução necessita de actualização constante, a tecnologia usada para implementação do cliente é JavaScript.

No seguimento desta ideia e da opção de adoptar um cliente de IM que use o protocolo XMPP, a interface terá um comportamento idêntico ao de um cliente de IM. Haverá um painel onde estarão listados os contactos, que terá ainda a possibilidade de ser minimizado. Será possível abrir uma janela para cada um dos contactos e esta, tal como o painel dos contactos, poderá ser minimizada. Na janela haverá um espaço para o utilizador escrever textos que podem servir como mensagens para outros utilizadores ou comandos para o servidor XMPP executar. Dentro de cada janela aparecerá a informação enviada pelo servidor e, no fim, serão adicionadas as acções disponibilizadas para executar sobre este tipo de informação; as acções podem passar por marcar um mensagem de correio electrónico como lida ou enviar um comentário relativo a uma foto do Facebook. As mensagens recebidas serão analisadas e, sempre que forem identificadas ligações

a conteúdos multimédia proveniente de outras plataformas, serão substituídas pelo conteúdo, de maneira transparente para o utilizador. A transferência de HTML nas mensagens, que foi abordada na Secção 4.4, possibilita uma apresentação melhor organizada e formatada ao utilizador. As *tags* HTML perigosas, como é o caso da *tag script*, serão desactivadas e aparecerão sob a forma de texto. Um protótipo com alguns dos conceitos da interface encontra-se na Figura 4.8.



(a) Janela de um contacto (b) Janela de uma fonte

Figura 4.8: Esboço da interface.

Como a interface será implementada em JavaScript, torna-se apenas necessário adicionar a biblioteca JavaScript à plataforma desejada, neste caso, às plataformas da Portugalmail implementadas em HTML e CSS. O último ponto de integração é a autenticação na aplicação uma vez que o cliente *Web* não disponibiliza a opção de *login* directamente ao utilizador. Ao efectuar *login* na plataforma da Portugalmail a aplicação efectuará o *login* automaticamente.

4.9 Conclusão

Os quatro pontos principais, abordados pela proposta para tornar a comunicação síncrona são: a arquitectura do sistema, a recolha de informação, a interacção com as fontes e a usabilidade do sistema. Neste capítulo foram definidas abordagens para esses pontos.

São usados vários componentes, alguns já completos outros com a necessidade de serem adaptados, e foram definidas metodologias de comunicação e formas de interacção entre eles. Através da utilização de algumas extensões do XMPP e da definição de novas estruturas alcança-se uma proposta que atinge os requisitos definidos.

Da necessidade de evolução do correio electrónico e da sua integração com outras fontes de informação, nasceu a abordagem definida neste capítulo que foi baptizada Wink[2].

[2]Wink (piscar) é um método de comunicação informal e que simboliza a rapidez e instantaneidade.

Capítulo 5

Preparação do Ambiente de Desenvolvimento

A proposta de solução passa por um sistema de vários componentes e que não é necessário desenvolver de raiz. Antes de começar a desenvolver o protótipo escolheram-se as aplicações existentes que mais se adequariam ao trabalho. Neste capítulo são abordadas essas escolhas e a configuração necessária para elas funcionarem em conjunto. Também são apontadas algumas características do ambiente de desenvolvimento.

5.1 Processo de Escolha das Aplicações de Base

Para arrancar com o desenvolvimento do protótipo foi necessário seleccionar algumas aplicações *open-source* entre as existentes. Nas próximas secções são apresentadas as escolhas e respectivas justificações.

5.1.1 Interface com o utilizador

No capítulo 3.3 é referido um estudo de clientes *Web* XMPP que foi efectuado para escolher um clientes. Como referido ao longo deste relatório a interface do projecto tem como base um cliente XMPP. Analisando esse estudo a opção escolhida foi o iJabBar[1].

Como foi definido no capítulo 4, a interface tem que fornecer todas as funcionalidades e componentes de um cliente de IM. Todos os clientes estudados têm funcionalidades como: visualizar a lista de contactos, alteração do estado, janelas criadas à medida que são recebidas mensagens de utilizadores diferentes e quando se recebe uma mensagem nova a janela é posta em evidencia,

[1] Página do código – http://code.google.com/p/ijab/

possibilidade de enviar mensagens a partir das janelas de cada utilizador, seleccionar um utilizador da lista de contactos e abrir a janela com esse utilizador. Mais específico do XMPP, todos os clientes permitem: fazer a ligação e autenticação no servidor através de *http-bind* e enviar todas as *stanzas*: *message*, *presence* e *iq*, necessárias para o funcionamento de um *instant messenger* normal.

O factor diferenciador é a integração com páginas *Web*, que é um dos requisitos apresentados. O cliente que melhor cumpre este requisito é o IJabBar, através do seu *design* concebido para a poupança de espaço e integração com outras aplicações *web*.

O iJabBar, ao contrário de alguns clientes como o JWChat, não fornece funcionalidades de registo e gestão de contas provenientes das *gateways*. Apesar de esta funcionalidade não estar implementada não será necessário alterar muito a sua estrutura para o fazer.

5.1.2 Comunicação e Integração com Servidores Existentes

Das *gateways* existentes foram escolhidas duas: a PyMSNt[2] e a J2J[3]. A PyMSNt é uma gateway entre o MSN e o XMPP e a J2J é uma gateway entre dois sistemas que usam o XMPP como protocolo.

As razões destas escolhas para o protótipo são a popularidade do MSN e a integração com uma rede social, como é o caso do Facebook, que usa o XMPP como protocolo de IM.

Apesar da escalabilidade das *gateways* ficar aquém do desejado, o objectivo destes componentes é mostrar que é possível a integração e aglomeração de várias contas de IM. Mesmo assim, um factor tido em conta aquando da escolha da PyMSNt e da J2J entre as *gateways* do mesmo protocolo foi a capacidade de escalar. Devido à restrição do tempo não foi implementada nenhuma *gateway* para a quantidade de utilizadores pretendidos, mas uma possibilidade de trabalho futuro será implementar uma, em Erlang, que é uma linguagem que tem como objectivo construir aplicações escaláveis.

5.1.3 Núcleo de Recolha de Informação e de Reencaminhamento de Dados

Após o estudo das diferentes implementações de servidores XMPP foi necessário escolher a mais adequada para expandir no projecto. Tendo em conta características como a abrangência de implementação, estado da comunidade e maturidade as opções foram restringidas a três: o Tigase Server, o Openfire e o Ejabberd. A escalabilidade e desempenho dos servidores são pontos fundamentais para a implementação final. Como tal, foram efectuados testes de desempenho e carga aos três servidores pré-seleccionados. Esta secção dá a conhecer os detalhes desses testes e respectivos resultados, sendo que no final é decidido o servidor a partir do qual se irá implementar o protótipo.

[2]Gateway MSN para o XMPP implementada em python – http://delx.net.au/projects/pymsnt/
[3]Gateway XMPP para o XMPP implementada em python – http://wiki.jrudevels.org/index.php/Eng:J2J

5.1.3.1 Ferramenta de Teste

Para efectuar os testes foi utilizada uma ferramenta *open-source* programada em Erlang com o nome Tsung. A versão do Tsung usada foi a 1.3.1.

Esta ferramenta pode ser usada para testar vários tipos de servidores, incluindo XMPP/Jabber, e pode ser distribuída por várias máquinas conseguindo assim simular centenas de milhar de utilizadores em simultâneo. Disponibiliza a funcionalidade de monitorização (CPU, memória e tráfego de rede) das máquinas através de agentes remotos (neste foi usado o Munin).

A Tsung permite efectuar as seguintes operações com o protocolo XMPP: autenticação, enviar informação de presença, enviar mensagens de registo, enviar mensagens de *chat* para utilizadores *online* ou *offline*, enviar pedidos de lista de contactos, enviar pedidos de *publish/subscribe* e efectuar operações sobre salas de conferência.

5.1.3.2 Características do Ambiente de Teste

Os testes foram efectuados nas mesmas condições para os 3 servidores. As versões dos servidores usadas foram Ejabberd 2.1.3, Openfire 3.6.4 e Tigase Server 4.3.1, ou seja, a versão mais recente de cada um aquando da realização deste estudo.

Dentro da rede local foram preparadas duas máquinas, uma com o servidor e outra com a ferramenta de testes. As características da máquina do servidor são: 960 652 kB de memória, um processador Intel(R) Pentium(R) 4 CPU 3.20 GHz e uma cache com 2048 kB de tamanho. As características da máquina da ferramenta de testes são: 443 852 kB de memória, um processador Intel(R) Pentium(R) 4 CPU 3.00 GHz e uma cache com 1024 kB de tamanho.

O sistema de gestão de base de dados usado em todos os servidores foi o PostgreSQL. As base de dados de cada servidor foram povoadas através de um *script* com 20 000 utilizadores, cada um com uma lista de 50 contactos.

5.1.3.3 Teste efectuado

Foi elaborado um conjunto de operações diversas, que são efectuadas em sequência por cada utilizador, separadas por algum tempo de espera. Estas operações, são definidas em XML como se pode ver detalhadamente na secção C.1 do Anexo C, são as seguintes:

- autentica-se
- envia a *stanza* de presença inicial
- requisita a lista de contactos
- envia a *stanza* de presença que o identifica como disponível para trocar mensagens com outros utilizadores
- envia a *stanza* de presença que o identifica como indisponível para trocar mensagens com outros utilizadores

- actualiza o estado para *Hello*
- adiciona um utilizador à lista de contactos e subscreve a sua presença
- envia novamente a *stanza* de presença que o identifica como disponível para trocar mensagens com outros utilizadores
- envia a *stanza* de presença final
- acaba a ligação

Para exercer uma boa carga sobre os servidores foi definido um teste com 10 min de duração e em que a frequência de chegada de utilizadores é um a cada 0.07 s.

5.1.3.4 Resultados Obtidos e Discussão

Tendo sido efectuados os testes, foram obtidos vários valores para diferentes operações e parâmetros. Na Tabela 5.1 podem ver-se os valores onde se verificam maiores discrepâncias e dos quais se podem tirar mais conclusões sobre as diferenças de performance dos servidores. Durante os testes foram obtidos mais dados que podem ser verificados no Anexo C.

Tabela 5.1: Valores que mais se destacaram durante o *benchmark*.

	Importância na decisão	**Ejabberd**	**c**[a]	**Openfire**	**c**[a]	**Tigase Server**	**c**[a]
Carga[b]	25%	0.53-0.28	5	1.15-0.24	2	1.13-0.50	3
Processador	25 %	~90 %	5	~160 %	3	~170 %	2
Memória Livre	15 %	~650 MB	4	~650 MB	4	~600 MB	3
Taxa de transferência do servidor para o cliente	5 %	693,25 kb/s (total: 44,93 MB)	3	496,80 kb/s (total: 31,06 MB)	3	869,58 kb/s (total: 52,46 MB)	4
Fase de autenticação	15 %	9.98 ms	5	31.94 ms	4	300.00 ms	1
Obtenção da lista de contactos	15 %	36,69 ms	4	22,51 ms	5	120,00 ms	1
Total[c]		4.6		3.35		2.2	

[a] Classificação de cada servidor no respectivo ponto. Escala varia entre 1 e 5.
[b] Os valores separados por hífen (-) representam o valor máximo e o valor mínimo obtidos durante o teste
[c] Valor obtido através da soma da multiplicação da classificação pela importância de cada linha.

Foram dados diferentes pesos aos vários parâmetros tendo em conta a sua importância. De destacar que a taxa de transferência tem um peso baixo uma vez que este parâmetro, no ambiente de testes, depende da quantidade de dados transferidos. Os parâmetros relativos a funcionalidades do servidor (autenticação e obtenção da lista de contactos) apenas não obtiveram mais peso uma vez que são funcionalidades que não serão das mais utilizadas por parte do cliente (uma vez por sessão).

Apesar de o teste ter sido limitado pelo *hardware* e não pelo *software* existem algumas conclusões que se podem retirar. O Tigase Server de um modo geral foi o servidor que obteve os piores resultados, sendo que o único ponto onde foi melhor foi na taxa de recepção. Este obteve

resultados bastante fracos na autenticação e na obtenção da lista de contactos. Analisando os parâmetros do Openfire verifica-se que foi o servidor mais rápido a obter a lista de contactos, mas por outro lado, foi o servidor que gerou menos carga sobre a máquina. O Ejabberd foi o servidor que se destacou mais obtendo os melhores valores em termos de carga e uso do processador, o que significa que suportaria mais utilizadores e mais operações que qualquer um dos outros na mesma máquina. Também se destacou na operação de autenticação obtendo o valor mais rápido.

Tendo apenas em conta estes valores, o Ejabberd será a melhor escolha uma vez que obteve a melhor pontuação, mas outros pontos influenciam a escolha final. O Tigase Server não tem a funcionalidade de integração com LDAP, que é o sistema de autenticação utilizado pela Portugalmail Comunicações S.A., e será necessário efectuar a sua implementação. O Openfire necessita de uma licença Oracle Coherence para funcionar em *cluster*, o que envolve custos adicionais para a empresa. Finalmente o Ejabberd é implementado na linguagem funcional Erlang [ADVW92] o que envolve riscos de conclusão do projecto com sucesso no tempo disponibilizado uma vez que é desconhecida nesta altura.

Após a ponderação sobre todos os parâmetros a escolha caiu sobre o Ejabberd. Apesar de aumentar os riscos da elaboração do projecto e de reduzir o tempo disponível para a implementação, devido à necessidade de aprendizagem da nova linguagem, este servidor obteve resultados nos testes que fazem pender a balança na sua escolha.

5.1.4 Conclusão

Das aplicações *open-source* existentes foram seleccionadas aquelas que melhor potenciam a realização do protótipo. As *gateways* escolhidas foram a PyMSNt para a integração com contas do protocolo MSN e a J2J para integração com contas de outros sistemas XMPP, como é o caso do Facebook e do Gtalk. O cliente *web* seleccionado para actuar como interface do sistema foi o iJabBar. Finalmente dos 3 servidores pré-seleccionados foi escolhido o Ejabberd.

5.2 Configurações

Nesta secção serão apresentados alguns pormenores relativos à preparação do ambiente de desenvolvimento, que estão directamente relacionados com os requisitos. Serão apresentadas as funcionalidades obtidas apenas com a junção das aplicações já existentes, e nos próximos capítulos serão abordados as funcionalidades adicionadas.

Foi instalado o Ejabberd 2.1.3 numa máquina com o Ubuntu 9.10 como sistema operativo. Na mesma máquina foi criada a base de dados em PostgreSQL que tem como objectivos, entre outros, guardar os utilizadores, as suas credenciais e guardar a lista de contactos de cada utilizador. Uma vez que o sistema de gestão de base de dados incorporado no Ejabberd, que é o Mnesia, tem vários problemas quando se processam grandes quantidades de dados [Hel06] foi decidido usar o PostgreSQL, uma vez que este SGBD é o usado preferencialmente na empresa.

O passo seguinte foi a configuração do servidor para autenticar os utilizadores por LDAP. Esta funcionalidade já se encontra implementada no Ejabberd e para a disponibilizar apenas é

necessário adicionar a informação necessária ao ficheiro de configuração do servidor. O servidor LDAP encontra-se numa máquina diferente, dentro da mesma rede interna.

As *gateways* foram instaladas na mesma máquina que o servidor e configuradas para poderem ser acedidas por ele. Elas poderiam ter sido instaladas em máquinas diferentes, mas, como se trata de um protótipo, foi tomada esta decisão uma vez que se pretendia apenas provar o seu funcionamento.

Finalmente, a aplicação cliente *Web* foi incorporada num servidor Apache e configurado para aceder ao servidor XMPP através do BOSH.

Nesta fase do protótipo é possível estabelecer ligação ao servidor através do cliente *Web* com credenciais do sistema da empresa. É possível iniciar conversas de *chat* com outros utilizadores, sendo que estes poderão ser do próprio sistema, do MSN ou outro sistema XMPP. O cliente já permite a actualização do estado do utilizador. Apesar do cliente suportar estas funcionalidades, não permite o registo nas *gateways* nem a adição de utilizadores logo, para testar estas funcionalidades não suportadas, foi necessário fazer estas operações noutro cliente que as disponibilizava.

5.3 Conclusão

Das tecnologias XMPP *Open-Source* existentes foram seleccionadas algumas para incluir no protótipo. Uma vez que era o único integrável com outra página *web*, e este era dos requisitos mais importantes na escolha da interface do projecto, foi escolhido o cliente iJabBar. Elegeram-se as *gateways* J2J e PyMSNt uma vez que se pretendia comunicar com o protocolo MSN e com o protocolo XMPP através de outras contas já existentes e porque entre as *gateways* existentes estas são as mais escaláveis. O servidor, como elemento nuclear do sistema, recebeu atenção adicional e foram efectuados testes de desenpenho. Através dos resultados obtidos e das restantes características levadas em conta, optou-se pelo Ejabberd. Finalmente, ligaram-se e configuraram-se todos os elementos, inclusive os da empresa, como o servidor LDAP e base de dados do Horde, obtendo-se imediatamente algumas das funcionalidades pretendidas.

Capítulo 6

Desenvolvimento do Protótipo de Comunicação em Tempo Real

Neste capítulo apresenta-se a arquitectura do protótipo e todas as alterações efectuadas sobre o cliente e servidor XMPP para implementar as funcionalidades especificadas na proposta de solução. Na última secção são enumeradas algumas conclusões sobre as funcionalidades implementadas e é avaliado se estas preenchem os requisitos apresentados.

6.1 Introdução

O segundo passo para a criação do protótipo, depois da preparação do ambiente de desenvolvimento, é a implementação das funcionalidades adicionais para colocar o sistema de acordo com as especificações da proposta de solução. As alterações são implementadas na interface e servidor uma vez que o funcionamento dos outros elementos já desempenham a função pretendida (como foi referido na Secção 4.4).

6.2 Arquitectura

Os elementos que necessitaram de ser alterados para implementar o conceito definido no capítulo 4 foram o servidor XMPP e o cliente XMPP. Na Figura 6.1 apresenta-se a arquitectura dos componentes implementados e o essencial dos componentes que já estavam implementados de raiz para entender o seu funcionamento.

Os componentes implementados no servidor foram vários módulos. Os módulos são: o mod_mail, mod_api_twitter, o mod_wink e o mod_roster_sybil. O mod_mail recolhe informação e realiza acções sobre o servidor de IMAP, o mod_api_twitter tem a mesma função que o mod_mail mas a fonte é o Twitter, o mod_wink disponibiliza várias funcionalidades (desde devolver o número de *mails* por ler até verificar se os dados de autenticação do MSN são válidos) e o

mod_roster_sybil é o módulo desenvolvido para substituir o módulo de gestão de lista de contactos que o Ejabberd tem por omissão. A estrutura interna dos módulos, apesar de não serem usados os mesmos nomes, coaduna-se com a estrutura definida no diagrama da Figura 4.4.

Na Figura 6.1 optou-se por não ilustrar a ligação entre os módulos e os restantes componentes do servidor para o diagrama não ficar confuso. Os módulos têm acesso aos métodos disponibilizados pelos componentes do Ejabberd: o ejabberd_router funciona como o principal reencaminhador de pacotes do servidor; se os pacotes tiverem como destino sistemas externos estes são reencaminhados do ejabberd_router para o S2S_manager que os enviará para o sistema indicado pelo domínio do destinatário. O ejabberd_odbc é usado para efectuar interrogações à base de dados.

Os módulos herdam os *behaviours* gen_mod e gen_server. O *behaviour* gen_mod define que quando o servidor inicia o módulo também é iniciado, e quando o servidor pára, o módulo também pára. O *behaviour* gen_server faz com que o módulo se adicione à tabela de encaminhamento do servidor com um determinado subdomínio, reencaminhando mensagens com esse subdomínio para o módulo. Outra maneira de definir o conportamento dos módulos é adicionando *hooks* disponibilizados pelo Ejabberd; um exemplo é despoletar uma acção quando uma *stanza* do tipo *presence* é enviada por um utilizador.

O cliente segue a estrutura definida na secção 4.3.3. Os componentes implementados no cliente foram o ActionStanzasBuilder, o MessageBuilder, o StanzaListener e o BarButtonWithLabel, sendo que estes dois últimos são os únicos que são novidade em relação à proposta. O StanzaListener é um componente que capta todas as *stanzas* recebidas pela biblioteca XMPP e tem como objectivo principal identificar quando o cliente recebe um pedido de subscrição por parte de outros contactos e nessa situação enviar a *stanza* de confirmação de subscrição e efectuar também o pedido de subscrição a esse contacto; as subscrições têm que ser efectuadas nos dois sentidos para permitir a comunicação entre os dois extremos; o servidor ao processar estas subscrições vai actualizando a tabela da lista de contactos da sua base de dados e alterando as suas permissões à medida que são trocados os pedidos. O BarButtonWithLabel é um componente baseado no BarButton, que é uma classe que representa um botão adicionado à barra de *chat* e que direcciona o utilizador para a página por ele definida (a diferença entre estas classes é que enquanto o BarButton é um botão com uma imagem, o BarButtonWithLabel tem uma imagem e uma etiqueta).

6.3 Gestão de *Gateways*

Nesta secção descreve-se a implementação da gestão de contas das *gateways* na interface. A sequência de passos da função principal de uma *gateway* é a seguinte: registo, *login*, envio da lista de contactos da *gateway* para o cliente, comunicação entre contactos e *logout*. Na fase de registo o cliente envia para a *gateway*, via servidor XMPP, as credenciais da conta. O cliente XMPP e os módulos desenvolvidos as comunicações efectuadas directamente com a *gateway* seguem as especificações do XMPP: o registo é efectuado usando especificações da XEP-0077 [Sai09b], o *login* de acordo com a especificação da XEP-0115 [HSTK08] e a lista de contactos usando

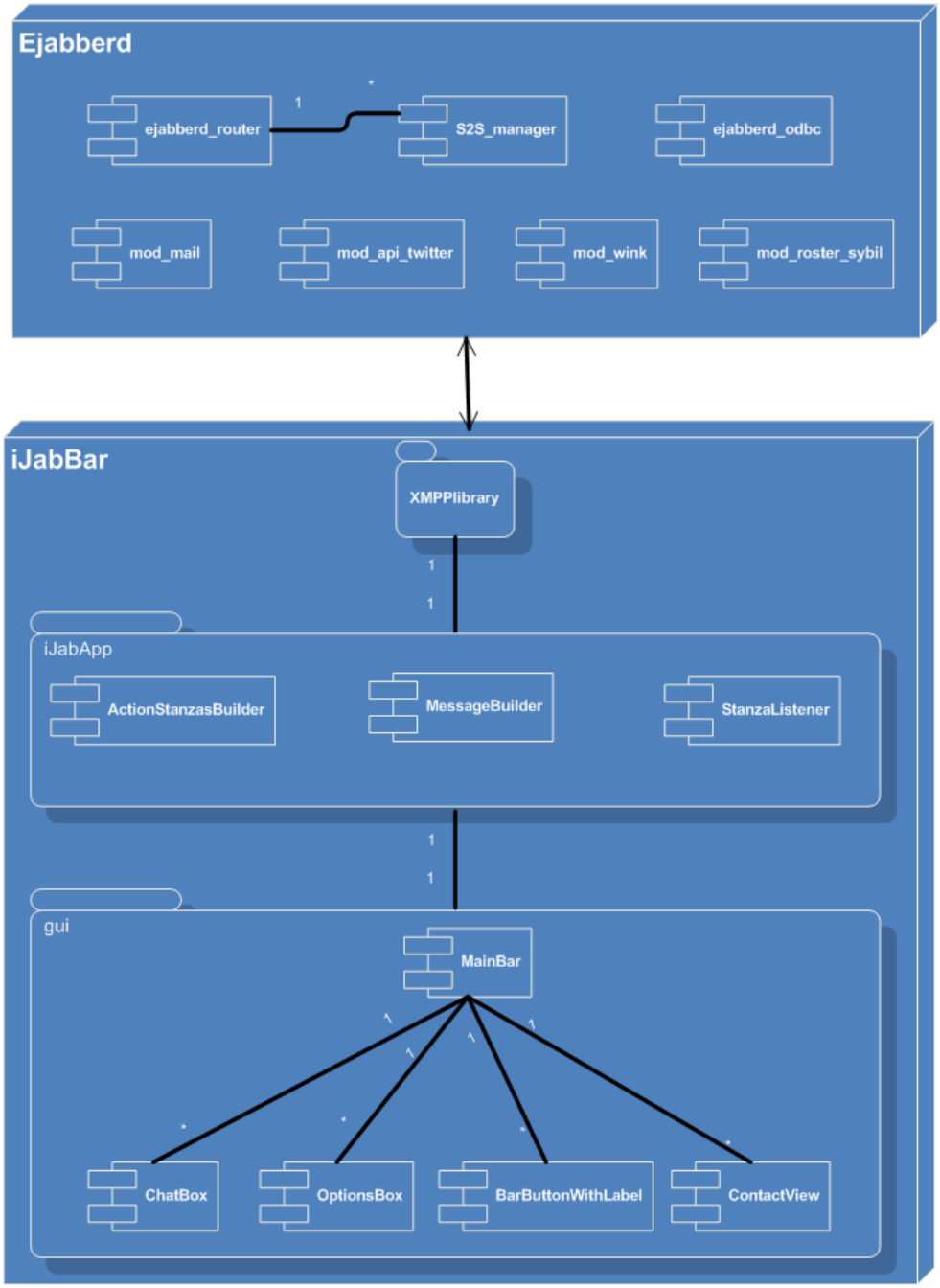


Figura 6.1: Arquitectura do protótipo.

as especificações da Instant Messaging and Presence do XMPP [Sai04], mais especificamente a secção da subscrição de contactos.

A interface para inserir as credenciais das contas é disponibilizada pela plataforma da empresa, trata-se de um simples formulário que na altura de submissão inicia o processo enunciado anteriormente. Na submissão é chamada a função registerService da *package* ActionStanzaBuilder que tem a seguinte actividade: envia a *stanza* de registo, se este envio for realizado com sucesso é efectuado o *login* e no caso de este ser validado é enviada pela *gateway* a lista de contactos que é analisada e subscrita pela StanzaListener, como explicado na secção 6.2. Com esta implementação torna-se possível a gestão das contas nas *gateways* e importação de contactos, apesar destes elementos da interface não serem implementados directamente no cliente as funcionalidades de gestão são disponibilizas pela sua biblioteca.

Como complemento à funcionalidade de registo, foi implementada uma nova funcionalidade de verificação da validade das credencias, para verificar se uma conta é válida antes de ser registada. Uma vez que apenas é possível verificar se uma conta é válida após fazer o registo na *gateway* e na validação não se quer adicionar imediatamente o utilizador à *gateway*, é usado outro método.

Envia-se o pedido do registo para o mod_wink com a *child* service para indicar em qual serviço se vai efectuar a validação. Um exemplo da stanza é o seguinte:

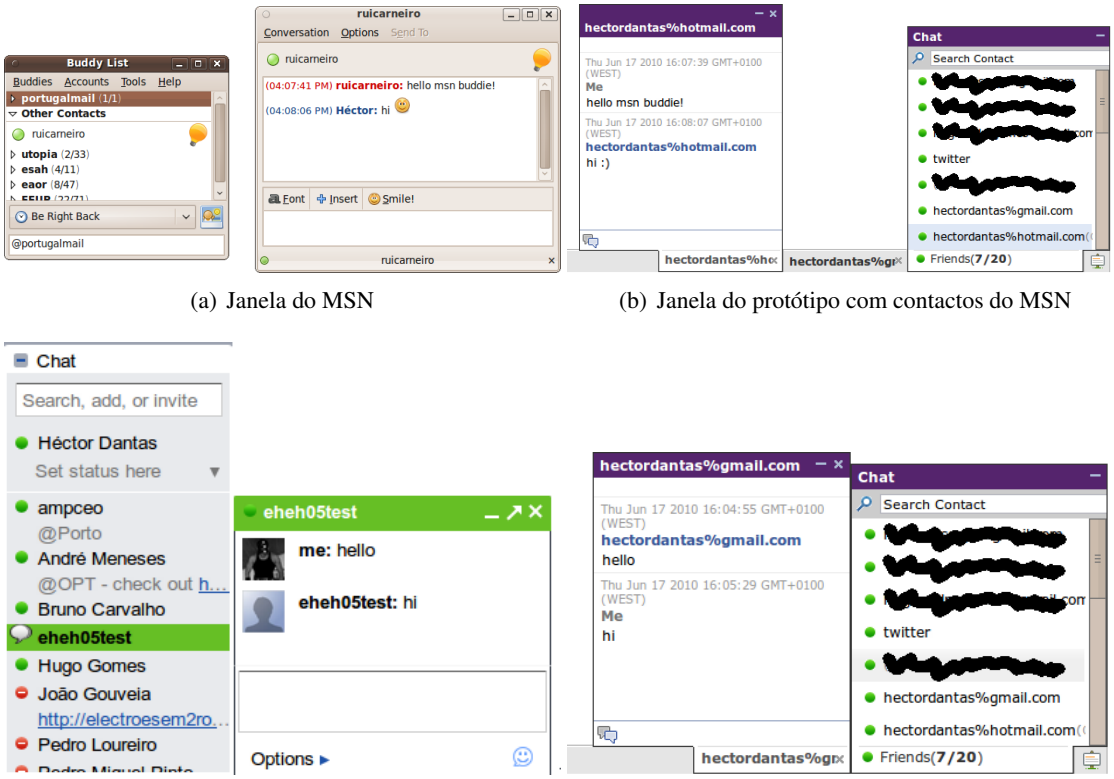
```
1 <iq xmlns="jabber:client" id="register_0" type="set" to="wink.portugalmail.pt">
2         <query xmlns="jabber:iq:register">
3                 <username>exemplo@msn.com</username>
4                 <password>pass123</password>
5         </query>
6         <service>msn</service>
7 </iq>
```

O módulo wink regista-se na *gateway* como o utilizador fictício user@wink.portugalmail.pt e recebe a confirmação da gateway. Dependendo do sucesso ou insucesso é enviada como resposta à *iq* outra *iq* com o atributo *type* igual a *result* ou *error*, respectivamente. Se o cliente XMPP receber a *stanza iq* com o *type* igual a *result*, ou seja, se o registo foi efectuado com sucesso, o cliente vai automaticamente efectuar uma tentativa de *login*. Para realizar esta acção envia uma *stanza* para o módulo Wink como, por exemplo:

```
1 <iq xmlns="jabber:client" id="servicedataget" to="wink.portugalmail.pt">
2         <query xmlns="service:data" />
3         <service>msn</service>
4 </iq>
```

Ao receber esta *stanza* o módulo efectua *login* na *gateway* como se fosse o utilizador fictício user@wink.portugalmail.pt, posteriormente recebe a informação de sucesso ou erro. No caso de sucesso o módulo Wink envia uma *stanza iq* com o atributo *type* igual a *result* como resposta à *iq* anteriormente enviada pelo cliente, no caso de erro envia uma *iq* com o atributo *type* igual a *error* com a mensagem de erro apropriada. Se a mensagem de erro for de conta inválida o cliente recebe esse erro, se receber a *iq* com o atributo *type* igual a *result* significa que a conta é válida.

No final desta fase o cliente XMPP passou a incorporar como funcionalidade a capacidade de se registar nas *gateways* e importar os seus contactos como se pode ver na Figura 6.2. Como trabalho futuro poder-se-á eliminar o passo intermédio da validação de conta entre o cliente e o modulo Wink, sendo que o cliente apenas envia uma *iq* a mandar validar, o modulo trata dos passos intermédios e depois envia a resposta final ao cliente.



(a) Janela do MSN

(b) Janela do protótipo com contactos do MSN

(c) Janela do Gmail

(d) Janela do protótipo com contactos do Gmail

Figura 6.2: Demonstração das funcionalidades da *gateway*.

6.4 Recolha e Visualização de Informação

Nesta secção é abordada a implementação do processo de recolha de informação das duas fontes que se decidiu usar, depois é demonstrado o método de envio e visualização da informação recolhida.

6.4.1 Processo de Recolha

No protótipo é efectuada a recolha de informação de duas fontes, o Twitter e o servidor de IMAP da empresa, uma externa e outra local respectivamente. A comunicação com o Twitter é realizada através de pedidos HTTP à sua API REST, a comunicação com o servidor IMAP é efectuada através de *sockets* TCP mas, fora estes métodos, o processo de recolha foi implementado

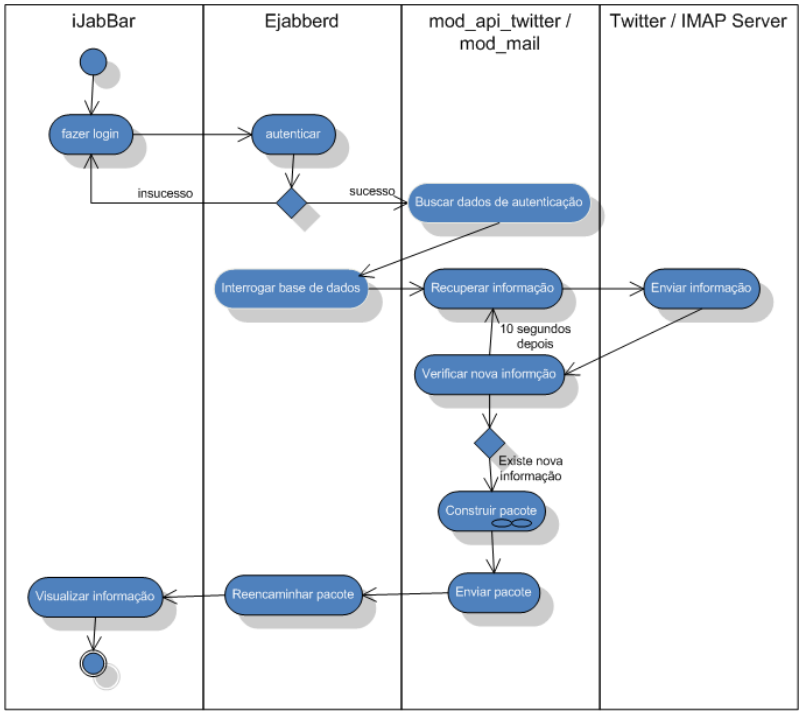


Figura 6.3: Diagrama de actividades da recolha e visualização de informação.

de igual forma para as duas fontes. Nesta secção é abordado este processo que se pode verificar no diagrama de actividade da Figura 6.3.

Quando se inicia o Ejabberd os módulos também se iniciam automaticamente, neste caso os módulos mod_api_twitter e mod_mail. Logo que um utilizador se autentica no servidor é activado o *hook* que alerta os módulos para o facto e estes criam imediatamente um processo concorrente para esse utilizador. Este processo concorrente interroga a base de dados do Horde, o local onde estão guardados os dados de acesso à fonte de cada utilizador, e com esses dados inicia a busca periódica de informação à fonte. Os processos concorrentes, criados para o utilizador, enviam o pedido para a sua fonte e são devolvidos os *tweets* da *wall* da conta respeitante aos dados obtidos da base de dados e as novas mensagens de correio electrónico. Nesta fase cada processo verifica o *timestamp* de todos os elementos recebidos e se este for maior que o *timestamp* da última verificação o elemento será enviado para o utilizador. O *timestamp* usado na verificação inicial é o da altura de *login* do utilizador (ou seja, qualquer informação posterior a esta ocasião é considerada nova) e depois da verificação inicial o *timestamp* usado é o respeitante à iteração anterior. Este método é usado para o utilizador ir recebendo só a informação nova.

Após obter a informação, e decidir se é nova ou não, o processo concorrente constrói uma *stanza* por cada elemento para enviar ao utilizador, segundo as especificações da Secção 4.5. As *stanzas* são enviadas para o destinatário através dos métodos de encaminhamento do Ejabberd e são apresentadas ao utilizador como se pode verificar na Figura 6.4. Um exemplo de uma *stanza* de correio electrónico será:

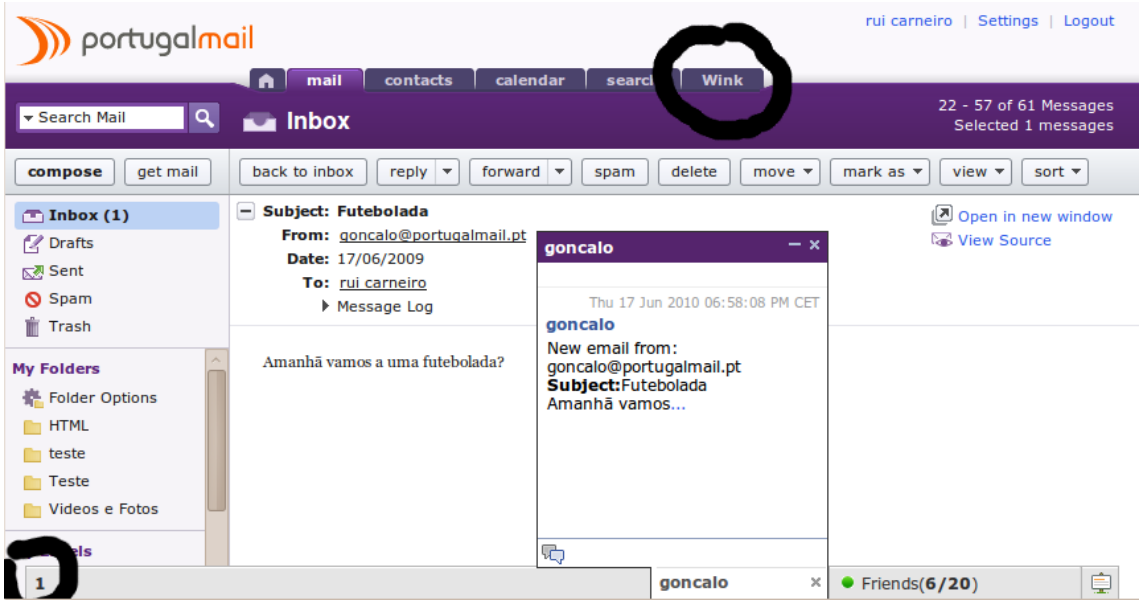


Figura 6.4: Demonstração da visualização de uma mensagem de correio electrónico no iJabBar.

```
1  <message from='goncalo@portugalmail.pt' to='ruicarneiro@portugalmail.pt' type='
       chat' source='internalIMAP'>
2    <body>
3      New mail from:
4      &lt;br/&gt;
5      goncalo@portugalmail.pt
6      &lt;br/&gt;
7      &lt;b&gt;Subject:&lt;/b&gt;Futebolada
8      &lt;br/&gt;
9      Amanh&atilde; vamos a uma futebolada?
10   </body>
11 </message>
```

Finalmente, quando o utilizador acaba a sessão no Ejabberd, é activado o *hook* respectivo e o módulos terminam os processos concorrentes que haviam criado para esse utilizador e a recolha acaba.

Concluindo, as funcionalidades dos módulos mencionadas nesta secção seguem as especificações e funcionamento definido na proposta e dependendo do período de recolha, definido pelo administrador do servidor, obtém-se a informação em tempos consideravelmente baixos (por exemplo, no protótipo, foi definido um período de 10 s).

6.4.2 Questões de Integração e Usabilidade na Visualização

Na proposta foi abordada a noção de agrupar contactos, foi introduzido o conceito de janela de fonte e janela de contacto. Esta noção levou à necessidade de implementar uma interface que permitisse os agrupamentos e a criação de uma tabela na base de dados do Ejabberd.

A interface implementada foi um simples formulário com 3 *combo boxes*: na primeira selecciona-se o contacto ao qual se pretende agrupar, na segunda a fonte da qual se pretende escolher um contacto e na terceira aparecerão os contactos filtrados pela fonte. Para ser de mais fácil compreensão fica o seguinte exemplo: um utilizador selecciona na primeira *combo box* um contacto da sua lista de contactos, na segunda selecciona como fonte o Twitter, na terceira irão aparecer todos os seus amigos do Twitter (ou seja, os amigos da conta que o utilizador terá previamente introduzido como sendo sua) ele secciona um desses amigos e submete essa informação. Ao submeter essa informação é chamado um método disponibilizado pelo cliente XMPP que envia uma *stanza* com a seguinte forma e conteúdo para o servidor, mais especificamente para o mod_wink:

```
<iq xmlns="jabber:client" id="addgroup" type="set" to="wink.portugalmail.pt">
        <query xmlns="gp:set:newgroup"/>
        <contact1>exemplo@portugalmail.pt</contact1>
        <contact2>usertwitter</contact2>
        <platform>twitter</platform2>
</iq>
```

O módulo wink quando recebe essa *stanza* extrai os dados dela e com ajuda dos métodos do ejabberd_odbc insere-os na base de dados do servidor, numa tabela previamente criada:

```
CREATE TABLE winkgroups (
    userjid text NOT NULL,
    contact1 text  NOT NULL,
    contact2 text NOT NULL,
    platformtype text NOT NULL
);
CREATE INDEX i_winkgroups_userjid ON winkgroups USING btree (userjid);
CREATE INDEX i_winkgroups_contact1 ON winkgroups USING btree (contact1);
```

Na Figura 6.3 existe um diagrama com a actividade estruturada *Construir pacote* onde são tomadas as decisões que influenciam o reencaminhamento da mensagem como vinda do contacto que representa a fonte ou de outro contacto normal. Essa actividade, esquematizada na Figura 6.5, traduz-se na utilização, por parte dos módulos que recolhem informação, dos métodos do ejabberd_odbc para interrogar a tabela winkgroups de forma a identificarem se a mensagem recebida está associada a algum contacto.

Ilustrando as declarações nesta secção com a Figura 6.6 das interfaces fica-se com a ideia mais concreta de uma história de utilização: um utilizador agrupa o contacto gonçalo@portugalmail.pt com o seu amigo do Twitter com o *username* mod_api3 como se pode ver na Figura 6.6(a), quando o servidor recolher informação nova da sua conta do Twitter verifica que tem várias mensagens novas, este verifica que as mensagens não são de utilizadores agrupados e envia-as com o remetente twitter@twitter.portugalmail.pt, como se pode verificar na Figura 6.6(b), mas uma é do utilizador mod_api3 que está agrupado com o gonçalo@portugalmail.pt, logo esta mensagem é enviada com o remetente gonçalo@portugalmail.pt, como se pode ver na Figura 6.6(c).

Outro conceito que se pretendia implementar era tornar as mensagens auto-contidas, ou seja, não ser necessário navegar para outras aplicações para visualizar algum conteúdo enviado nas mensagens. Com esse intuito foi criado o componente MessageBuilder no cliente XMPP. Ou seja,

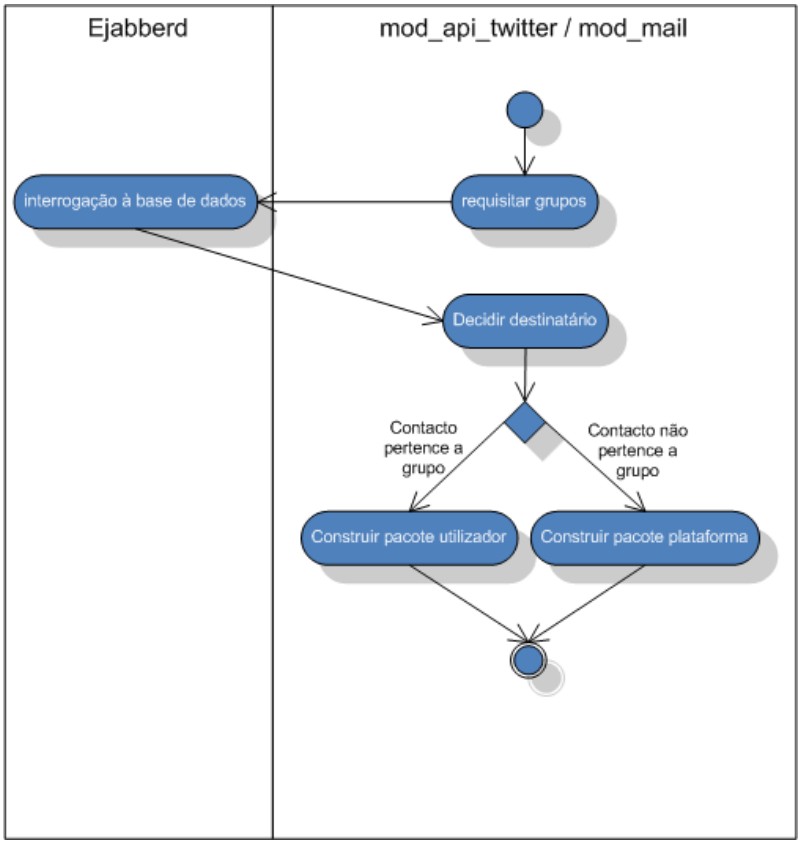


Figura 6.5: Diagrama de actividades da actividade estruturada ”Construir pacote”.

(a) Interface para agrupar contactos

(b) Janela de uma fonte

(c) Janela de um contacto

Figura 6.6: Demonstração da funcionalidade de ”Agrupar contactos”.

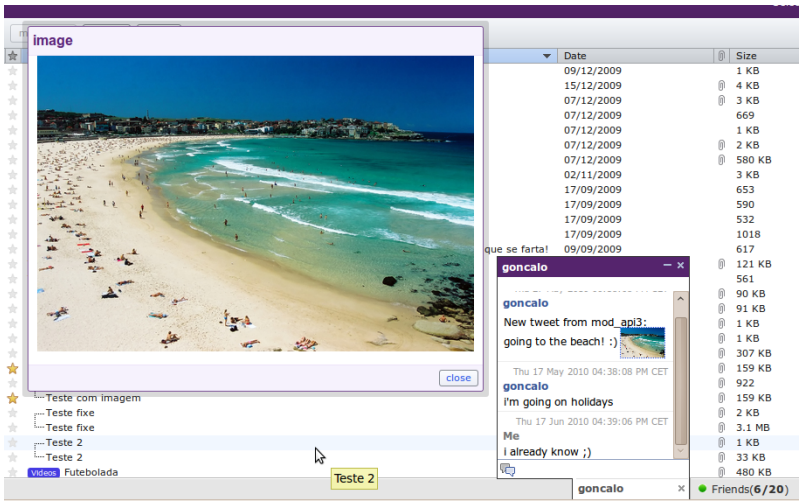


Figura 6.7: Demonstração da funcionalidade de ”Conteúdo *inline*”.

antes de apresentar a mensagem ao utilizador, a mensagem é transformada através de expressões regulares. São identificados as Uniform Resource Locators (URL) na mensagem e se forem de imagens ou vídeos do Youtube estas são substituídas por *thumbnails* sobre os quais é possível efectuar um clique e visualizar o conteúdo num *pop-up*, como se pode ver na Figura 6.7.

O protótipo foi implementado de acordo com as especificações, apenas ficando por implementar a funcionalidade de adicionar opções às mensagens recebidas, que seria efectuada no MessageBuilder. Mas facilmente se verifica que seria simplesmente adicionar mais código HTML ao fim da mensagem, sendo que as opções adicionadas dependeriam do atributo *source* da mensagem.

6.5 Acções Sobre a Fonte de Informação

Na proposta de solução foram abordados dois métodos de interagir com as fontes de informação: através de funções disponibilizadas pela interface que enviam uma *stanza iq* para o respectivo módulo que por sua vez executa a acção e através de *stanzas* do tipo *message* enviadas para o respectivo módulo, provenientes da janela do contacto que simboliza a fonte.

O primeiro método não foi implementado no protótipo mas rapidamente se percebe, através de alguns exemplos, que seria possível implementá-lo. Neste caso o código é inserido nas mensagens para criar o *pop-up* onde se visualizam os conteúdos como imagens e vídeos. Basicamente é inserido na mensagem um objecto "clicável" que chama uma função disponibilizada pelo cliente e que é construído da seguinte forma:

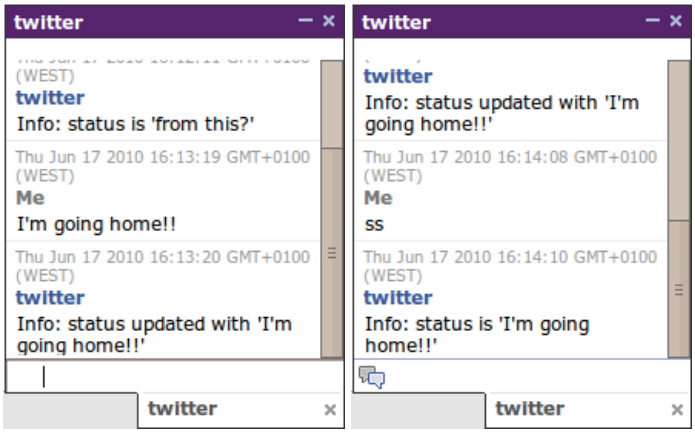
```
<a href=\"javascript:iJab.openRedBox(\'http://www.youtube.com/v/1234ABCD\',\'
    video\')\" title=\"Image\"><img src=\"http://img.youtube.com/vi/1234ABCD/2.
    jpg\" alt=\"window\" width=\"50px\" /></a>
```

Apenas seria necessário adicionar os botões à mensagem que chamariam as funções, sendo que as opções adicionadas dependeriam do atributo *source* e do atributo *role*. O cliente teria que saber as

funções disponibilizadas para cada tipo de mensagem de cada fonte. Finalmente a capacidade de uma função enviar *stanzas* do tipo *iq* para efectuar acções nos módulos é amplamente comprovada em várias funcionalidades.

O segundo método, que passa por enviar mensagens com comandos, consiste no utilizador escrever mensagens com uma sintaxe predefinida na janela de uma fonte, como se pode ver na Figura 6.8. Estes comandos podem-se dividir em duas categorias: comandos sobre o funcionamento do módulo e comandos de acções a executar na fonte. O conteúdo da *tag body* é analisada no módulo. Um exemplo de uma mensagem no formato XMPP é o seguinte:

```
1 <message from='exemplo@portugalmail.pt' to='twitter@twitter.portugalmail.pt' id
    ='msg_1'>
2   <body>mode on</body>
3 </message>
```



(a) Actualizar estado do Twitter (b) Ver o estado no Twitter

Figura 6.8: Demonstração da funcionalidade de comandos sobre os módulos.

Os comandos sobre o funcionamento do módulo implementados no mod_api_twitter e no mod_mail foram: *mode on*, *mode off* e *mode auto* [Pro10b]. Como foi abordado anteriormente, estes módulos criam um processo concorrente para recolher a informação de cada utilizador e o identificador deste processo é guardado numa tabela de *routing* associado à JID desse utilizador. Ao receber uma mensagem com o comando *mode on* proveniente de um utilizador, o módulo acede à tabela e depois de ter em sua posse o identificador do processo envia-lhe uma mensagem para este começar a recolher informação. Esta mensagem é diferente das mensagens XMPP: é uma forma de comunicação entre processos da linguagem Erlang. O fluxo no servidor é o mesmo recebendo *mode off* e *mode auto*, com a diferença da mensagem enviada para o processo concorrente ser diferente e ter acção diferente sobre este: o *mode off* suspende a recolha de informação e põe o processo em modo de espera, o *mode auto* põe o processo em modo de espera ou a recolher informação dependendo do estado do utilizador (chat-recolhe; away,dnd,xa-espera). De forma a manter persistência nas opções do utilizador, o módulo guarda o seu estado para cada utilizador. A tabela seguinte foi criada na base de dados do Ejabberd para esse efeito:

```
1 CREATE TYPE status AS ENUM ('on', 'off', 'auto');
2 CREATE TABLE winkstatus
3 (
4   username text NOT NULL,
5   platform text NOT NULL,
6   mode status NOT NULL
7 )
8 CREATE INDEX i_api_username  ON api USING btree (username);
```

Quando um módulo inicia o processo concorrente ele começa por obter o estado da tabela, para que o processo inicie da forma que o utilizador o programou.

Os comandos para executar na fonte apenas foram implementados no módulo mod_api_twitter, por questões relativas a gestão de tempo, e foram os seguintes: <*message*>, *ss*, *d* <*user*> <*message*>. O comando <*message*>, que na prática é apenas texto livre que não coincida com outro comando, é usado para efectuar um *tweet* na sua parede (*wall*). O comando *ss* é usado para obter o ultimo *tweet* que o utilizador efectuou na sua parede. O comando *d* <*user*> <*message*> é usado para enviar uma mensagem directa a um utilizador do twitter. O módulo mod_api_twitter ao receber uma *stanza message* que tenha um destes comandos interroga a base de dados Horde para obter os dados de acesso do utilizador à sua conta do Twitter e posteriormente faz a chamada à API REST do Twitter com a acção desejada. Apesar deste método de comandos não ser o que tem maior usabilidade serve apenas para ilustrar a maneira como se pode actuar sobre as fontes a partir do cliente XMPP.

Com a implementação destas funcionalidades ficam demonstradas as acções sobre as fontes, enviadas pelo cliente e executadas pelo servidor.

6.6 Conclusão

Apesar de não terem sido implementados todos os requisitos, as funcionalidades que foram implementadas seguem as especificações. As que não foram implementadas seriam perfeitamente possíveis de implementar e foi apresentada a justificação para esse facto. Há inúmeros pormenores de usabilidade que podiam ser melhorados, mas o funcionamento global pretendido foi atingido. Com a excepção de alguns pontos de integração da interface e do preenchimento da lista de contactos de cada utilizador, que são abordados no capítulo seguinte, os requisitos funcionais são preenchidos pelo protótipo.

Capítulo 7

Integração do Protótipo com a Plataforma de Correio Electrónico

Este capítulo aborda a integração do protótipo com a plataforma de correio electrónico existente na Portugalmail. É feita uma breve introdução para situar o leitor, depois são mencionados os pormenores da integração da interface na plataforma de correio electrónico. Aborda-se a implementação do módulo do servidor responsável por enviar a lista de contactos para cada utilizador, enumeram-se algumas funcionalidades desenvolvidas para fundir as duas aplicações e tiram-se algumas conclusões sobre esta integração.

7.1 Introdução

Já foi mencionada a integração ao nível da autenticação no protótipo que não envolveu qualquer tipo de implementação, sendo apenas uma questão de configurações (ver Secção 5.2). Neste capítulo é abordada a integração que levou à implementação de certas funcionalidades que serão descritas seguidamente.

No que respeita à lista de contactos e sua gestão apenas se mencionou a subscrição de contactos das *gateways*, mas também se pretende a importação de elementos provenientes do gestor de contactos da plataforma de correio electrónico.

Grande parte das redes sociais e plataformas de *micro-blogging* entre outras possíveis fontes, permitem que o utilizador actualize o seu estado. Esta situação verifica-se inclusivamente no gestor de contactos da plataforma de correio electrónico, o que potenciou o aparecimento de uma nova funcionalidade.

7.2 Integração ao Nível da Interface

Foi criado um módulo na plataforma de correio electrónico para servir como contentor da interface. Para o utilizador isto traduz-se numa nova *tab* na interface da plataforma como se pode ver na Figura 7.3 da página 57. Essa *tab* apresenta ao utilizador a interface de agrupamento de contactos, que é a única desenvolvida além do cliente *Web* XMPP; de notar que a interface de adição de dados de acesso às fontes de informação é disponibilizada pelo gestor de contactos da plataforma de correio electrónico. A biblioteca JavaScript, e respectivos ficheiros, que implementa o iJabBar, com as alterações efectuadas, foi adicionada à estrutura de pastas do módulo criado. Alteraram-se os cabeçalhos da plataforma de correio electrónico de maneira a que a biblioteca fosse inserida em todas as páginas. Desta forma a interface do Wink encontra-se disponível em todas as páginas da plataforma de correio electrónico.

Após ter a interface disponível na plataforma, implementou-se a sincronização com o sistema de autenticação. Foi criada uma função JavaScript que, através de um pedido de AJAX à API da plataforma, identifica o utilizador que está autenticado e efectua o *login* no cliente. Se a sessão no cliente já tiver sido iniciada, este continua o seu funcionamento normalmente. Foi implementada a função da API da plataforma de correio electrónico que fornece informação de autenticação do utilizador em sessão no formato JSON.

Quanto à sincronização do *logout* foi simplesmente necessário chamar a função que termina a sessão no cliente na altura em que o utilizador faz *logout* na plataforma de correio electrónico.

7.3 Lista de Contactos Importada do Gestor de Contactos

Na Secção 4.7 é explicado como são adicionados outros utilizadores do sistema à lista de contactos. Nesta secção será exemplificado como se efectuou essa agregação de contactos das *gateways* e do gestor de contactos da plataforma existente na Portugalmail.

O componente que vem por omissão no Ejabberd para fazer a gestão da lista de contactos, quando recebe a requisição da lista de contactos do utilizador faz uma interrogação à base de dados do servidor XMPP e devolve a lista pedida com base nos valores devolvidos pela interrogação. Os contactos provenientes das *gateways* são guardados na base de dados do servidor, mas os contactos provenientes do gestor de contactos encontram-se noutra base de dados que é gerida independentemente do servidor XMPP. Poder-se-iam importar os contactos dessa base de dados para a base de dados do servidor mas envolveria grandes esforços para realizar a sua sincronização e por isso foi decidido adaptar o componente do Ejabberd, criando o mod_roster_sybil. Quando o utilizador pede a lista de contactos, este novo módulo além de devolver os contactos que extraiu da base de dados do servidor devolve também os contactos que extraiu da base de dados do gestor de contactos.

Finalmente, ainda relativo a preenchimento da lista de contactos, foi adicionada ao cliente *Web* a capacidade de adicionar contactos seguindo as especificações do RFC 3921 [Sai04]. Esta funcionalidade é necessária para adicionar à lista de contactos do utilizador o contacto relativo

Figura 7.1: Demonstração da actualização de estado no cliente *Web*.

a fontes de informação, por exemplo, adicionar o contacto twitter@twitter.portugalmail.pt. Esta funcionalidade efectua automaticamente o pedido de ligação e depois o pedido de *subscribe*, o que faz com que este contacto acabe sendo adicionado à base de dados do servidor XMPP. O cliente XMPP adiciona automaticamente, através desta função, os contactos das fontes caso estes ainda não existam na lista de contactos.

7.4 Funcionalidades Adicionadas

Uma das funcionalidades de partilha de informação que se decidiu implementar foi a actualização de estado do gestor de contactos com base no estado definido no cliente *Web*. Como se pode verificar na Figura 7.1, a interface dispõe de um menu onde o utilizador pode actualizar a sua mensagem de estado. Aproveitou-se esta funcionalidade para, além de actualizar no cliente, também actualizar a mensagem de estado do gestor de contactos.

Ao actualizar o estado é enviada a seguinte *stanza* para o servidor:

```
1 <presence xml:lang='en'>
2   <show>dnd</show>
3   <status>a ver o mundial</status>
4 </presence>
```

Adicionou-se o *hook*, ao mod_wink, que é activado quando é enviada uma *stanza* do tipo *presence* para o servidor. O mod_wink ao receber estas *stanzas* verifica se trazem incluída a *tag status* e caso isso aconteça actualiza o campo da base de dados do Horde onde é guardado o estado do utilizador pelo gestor de contacto.

Com o objectivo de reaproveitar as funcionalidades do cliente desenvolveu-se um método que devolve a presença de um utilizador, informação se ele se encontra ligado na plataforma ou não. Podemos ver na Figura 7.2 um exemplo da interface implementada para testar o método. O gestor de contactos permite a visualização da página de informação de cada amigo mas este método serve

unicamente para indicar nessa página se o amigo se encontra ligado e não ser preciso procurar na lista de contactos do cliente *Web*.

(a) Utilizador *online*

(b) Utilizador *offline*

Figura 7.2: Demonstração da presença de um utilizador.

O método devolve o estado do amigo a partir da informação da lista de contactos, se ele se encontra *online* é devolvido verdadeiro, se este se encontra *offline* é devolvido falso.

Uma funcionalidade implementada em que se verifica a integração com o servidor de correio electrónico é a funcionalidade das mensagens por ler. Foi adicionado um botão à barra do cliente através da classe BarButtonWithLabel onde se encontra a informação de quantas mensagens de correio electrónico o utilizador tem por ler, como se pode verificar no canto inferior esquerdo da Figura 7.3.

O método implementado tem um período configurável e recolhe ciclicamente o número de mensagens por ler no servidor de IMAP. Para tal envia uma stanza para o mod_wink com a seguinte forma:

```
1 <iq xmlns="jabber:client" id="getnewmessages" type="get" to="wink.portugalmail.
    pt">
2         <query xmlns="pm:get:newmessages"/>
3 </iq>
```

O mod_wink interroga o servidor IMAP e, após receber a resposta, envia a iq *result* para o cliente onde dentro da *tag query* vai o número de mensagens novas, ou então em caso de insucesso é enviada a resposta iq de erro. A *stanza result* tem a seguinte forma:

```
1 <iq xmlns="jabber:client" id="getnewmessages" type="result" to="
    exemplo@portugalmail.pt">
2         <query xmlns="pm:get:newmessages">21</query>
3 </iq>
```

7.5 Conclusão

Foi implementado com sucesso o módulo de gestão de contactos híbrido, com contactos provenientes de duas fontes diferentes. Este novo módulo além de fornecer a lista de contactos provenientes de duas fontes diferentes também permite a subscrição de contactos, assim como fornece

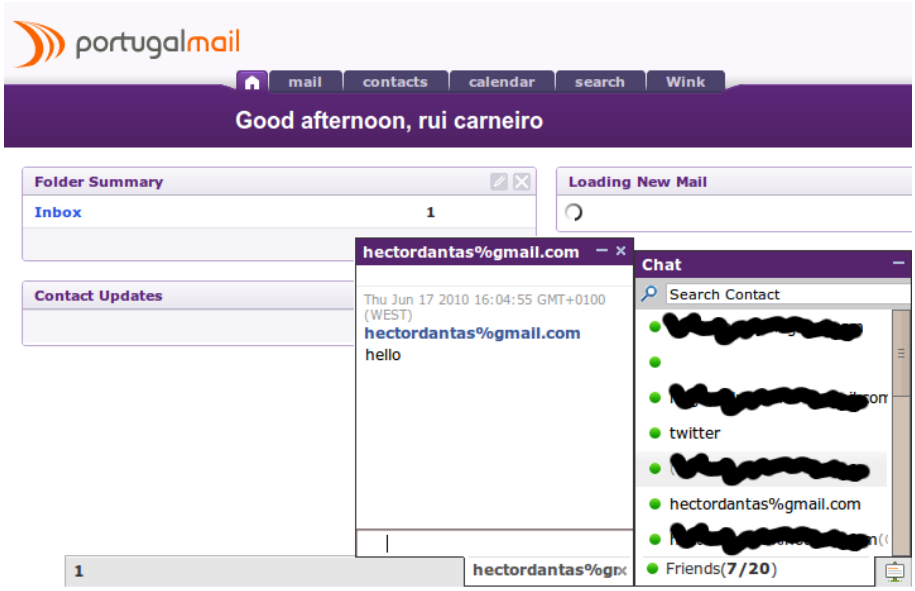


Figura 7.3: Demonstração da informação de mensagens novas no cliente *Web*.

as permissões necessárias para o utilizador visualizar a presença dos seus contactos.

Embora ainda existam pormenores relativos a segurança a serem melhorados, a integração do cliente *Web* com a plataforma de correio electrónico foi realizada com sucesso. A interface passou a ser usada em sincronia com a plataforma de correio electrónico no que diz respeito a autenticações no sistema.

Foram implementadas algumas funcionalidades de partilha de informação aproveitando as potencialidades do XMPP, demonstrando-se, mais uma vez, a interacção com fontes de informação; neste caso com o servidor de IMAP e com o gestor de contactos da plataforma de correio electrónico.

Capítulo 8

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões finais da dissertação, incluindo o sucesso da solução e o protótipo implementado, as inovações, contribuições e limitações da solução. Por fim são enumeradas algumas possibilidades de trabalho futuro.

8.1 Conclusões

Foi elaborada uma proposta com base nos objectivos designados pela Portugalmail – Comunicações, S.A. enumerados na Secção 1.3. Apesar de deixar em abstracto alguns pormenores devido ao alargado espectro de fontes que podem ser consideradas, a proposta de solução abrange com sucesso todos os pontos especificados nesses objectivos. Os objectivos nucleares, como é o caso da evolução do correio electrónico para modelos em tempo real, a agregação de informação e interacção com as fontes desta e finalmente a integração com a plataforma de correio electrónico da empresa, foram totalmente contemplados pela solução. Foram abrangidos outros requisitos que surgiram ao longo da investigação e que se acharam interessantes como é o caso do agrupamento de contactos e a visualização de conteúdos sem mudar de plataforma.

Com base na proposta implementou-se um sistema que cumpre as suas especificações. As funcionalidades adicionas ao projecto foram as necessárias para provar que a implementação da proposta é possível. Apenas os requisitos não funcionais não foram completamente implementados e provados no protótipo, nomeadamente a segurança e a escalabilidade.

Apesar não ter sido provada a escalabilidade do protótipo foram tomadas algumas decisões para que este fosse o mais escalável possível. Estas decisões foram baseadas num *benchmark* efectuado ao elemento principal do sistema que é o servidor XMPP; com base nestes testes foi escolhido o Ejabberd. O Ejabberd é implementado em Erlang, que é uma linguagem de programação concorrente [ADVW92].

A integração com a plataforma de correio electrónico foi atingida embora com alguns problemas de segurança que poderiam ser resolvidos com uma implementação mais demorada. A

autenticação no cliente *Web* não é realizada da forma mais adequada, sendo apresentado em trabalho futuro um método mais seguro.

Na actualidade estão a surgir ferramentas que ainda se encontram em fase de desenvolvimento e que tentam desenvolver uma nova abordagem à comunicação na *Web*, como é o caso do Mozilla Raindrop e o Google Wave. Esta dissertação é uma tentativa de apresentar uma nova abordagem integrável com uma plataforma existente: neste caso a plataforma de correio electrónico da Portugalmail.

Outro dos aspectos inovadores da proposta assenta no aproveitamento de tecnologia XMPP, como é o caso do servidor e do cliente de *Instant Messaging* para servirem como ponto de partida para o desenvolvimento, nomeadamente nas características do cliente *Web* para servir como interface do protótipo. De facto, efectuou-se uma extensão ao protocolo XMPP para permitir a implementação dos requisitos.

Uma aplicação desenvolvida num espaço de tempo tão curto tem as suas limitações. Embora tenha sido definido o fluxo de dados para a comunicação com todo o tipo de fontes, estes métodos de comunicação não puderam ser testados para todos os tipos de fontes no protótipo. Especialmente a transferência de informação em binário, como *streams* de vídeo e áudio não foi abordada convenientemente. A falta de implementação de *gateways* escaláveis é outra limitação; apesar de ser implementável com as tecnologias existentes não foi abordada devido ao limite temporal do projecto. Outro ponto em falta nesta dissertação foi um estudo final de usabilidade que permitisse avaliar a reacção dos utilizadores a esta nova abordagem.

8.2 Trabalho Futuro

Um ponto que não foi abordado na proposta foi o armazenamento das mensagens para posterior visualização e interacção. Apenas foi elaborada para comunicação em tempo real, descartando a informação enviada quando o utilizador não se encontra *online*. Um passo em frente no projecto seria a implementação deste armazenamento, visualização e gestão.

Além dos filtros dos contactos agrupáveis poderiam ser incluídos novos filtros de maneira ao utilizador ser alertado mais explicitamente das mensagens mais importantes em detrimento das menos relevantes, à imagem dos filtros implementados no Mozilla Raindrop.

Relativamente a filtros, seria interessante aglomerar ou esconder do utilizador mensagens duplicadas vindas de várias fontes. Por exemplo, ao receber um comentário efectuado a uma foto na sua conta do Facebook não receber também a notificação que foi enviada para o seu *email* a anunciar que a sua foto foi comentada.

A falta de *gateways* escaláveis foi mencionada em alguns pontos desta dissertação; a implementação destas *gateways* seria um passo a considerar no futuro uma vez que a aglomeração de contas de IM é um ponto importante. As *gateways* poderiam ser programadas numa linguagem que permitisse o tipo de concorrência necessária, por exemplo, Erlang.

Finalmente, a última sugestão passa por implementar um novo método de autenticação. Ao autenticar-se na plataforma de correio electrónico, o servidor enviaria a informação automaticamente para o servidor XMPP, sem necessidade dos dados passarem pelo cliente *Web*, ou seja, o servidor da plataforma iniciaria o processo de autenticação no servidor XMPP sendo que o cliente posteriormente dava seguimento à sessão.

Referências

[ADVW92] J. L. Armstrong, B. O. Däcker, S. R. Virding e M. C. Williams. Implementing a functional language for highly parallel real time applications. Technical report, Computer Science Laboratory, Ellemtel Utvecklings AB, 1992. Publicado em 1 de Abril de 1992 na SETSS 92, disponível em http://www.erlang.se/publications/implem.pdf.

[AMS10] LLC (AMS) Association Management Solutions. Internet engineering task force, 2010. Disponível em http://www.ietf.org/rfc.html, acedido a última vez em 23 de Janeiro de 2010.

[Dan10] Héctor Dantas. Evolução de correio electrónico para modelos de comunicação em tempo real – monografia para PDIS, 2010. Disponível em http://aitchdee.com/diswiki/files/HectorDantas_Monografia.pdf, acedido a última vez em 28 de Junho de 2010.

[Dom09] Gaston Dombiak. Ignite realtime: Ignite realtime blog: Clustering plugin for openfire is now open source, 2009. Publicado em 10 de Novembro de 2009, disponível em http://www.igniterealtime.org/community/blogs/ignite/2009/11/10/clustering-plugin-for-openfire-is-now-open-source, acedido a última vez em 28 de Janeiro de 2010.

[Fer09] Andrés Ferraté. *Google Wave: Up and Running: Rough Cuts Version*. 2009. Publicado em Novembro de 2009 na O'Reilly Media.

[Fou10a] XMPP Standards Foundation. XMPP extensions, 2010. Disponível em http://xmpp.org/extensions/, acedido a última vez em 28 de Junho de 2010.

[Fou10b] XMPP Standards Foundation. XMPP RFCs, 2010. Disponível em http://xmpp.org/rfcs/, acedido a última vez em 31 de Maio de 2010.

[Fou10c] XMPP Standards Foundation. XMPP standards foundation, 2010. Disponível em http://xmpp.org/, acedido a última vez em 24 de Junho de 2010.

[Goo10a] Google. Google wave API overview - google wave API - google code, 2010. Disponível em http://code.google.com/apis/wave/guide.html, acedido a última vez em 28 de Janeiro de 2010.

[Goo10b] Google. Wave extensions - google wave API - google code, 2010. Disponível em http://code.google.com/apis/wave/extensions/, acedido a última vez em 28 de Janeiro de 2010.

[Hef07] Artur Hefczyc. Tigase.org | open source and free (GPLv3) Jabber/XMPP server, 2007. Publicado em 18 de Outubro de 2007, disponível em http://www.tigase.org/node?page=20, acedido a última vez em 25 de Junho de 2010.

[Hel06] Emil Hellman. *Evaluation of Database Management Systems for Erlang*. PhD thesis, IT University of Gothenburg, Sweden, June 2006.

[HSTK08] Joe Hildebrand, Peter Saint-Andre, Remko Tronçon e Jacek Konieczny. XEP-0115: entity capabilities, 2008. Actualizado em 26 de Fevereiro de 2008, disponível em http://xmpp.org/extensions/xep-0115.html, acedido a última vez em 3 de Junho de 2010.

[Jon09] M. Tim Jones. Meet the extensible messaging and presence protocol (XMPP). 2009. Publicado em 15 de Setembro de 2009 na IBM DevelopersWork, disponível em http://www.ibm.com/developerworks/library/x-xmppintro/, acedido a última vez em 28 de Janeiro de 2010.

[LBS+09] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan e Joe Hildebrand. XEP-0166: jingle, 2009. Actualizado em 23 de Dezembro de 2009, disponível em http://xmpp.org/extensions/xep-0166.html, acedido a última vez em 9 de Junho de 2010.

[LT10] Soren Lassen e Sam Thorogood. Google wave federation architecture (Google wave federation protocol). 2010. Publicado em 1 de Maio de 2009 na Google Wave Federation Protocol, Disponível em http://wave-protocol.googlecode.com/hg/whitepapers/google-wave-architecture/google-wave-architecture.html, acedido a última vez em 20 de Junho de 2010.

[Moz10] Mozilla. Raindrop - MozillaWiki. https://wiki.mozilla.org/Raindrop, 2010. Disponível em https://wiki.mozilla.org/Raindrop, acedido a última vez em 25 de Junho de 2010.

[PC10a] S.A. Portugalmail Comunicações. Blogs gratuitos | portugalmail, 2010. Disponível em http://www.portugalmail.net/servicos/blog/blogs-gratuitos, acedido a última vez em 28 de Janeiro de 2010.

[PC10b] S.A. Portugalmail Comunicações. Business email | portugalmail, 2010. Disponível em http://www.portugalmail.net/servicos/email/email-profissional, acedido a última vez em 28 de Janeiro de 2010.

[PC10c] S.A. Portugalmail Comunicações. Email gratuito | portugalmail, 2010. Disponível em http://www.portugalmail.net/servicos/email/email-gratuito, acedido a última vez em 28 de Janeiro de 2010.

[PC10d] S.A. Portugalmail Comunicações. Portal blog ASP | portugalmail, 2010. Disponível em http://www.portugalmail.net/servicos/blog/portal-blog-asp, acedido a última vez em 28 de Janeiro de 2010.

[PC10e] S.A. Portugalmail Comunicações. Portal email ASP | portugalmail, 2010. Disponível em http://www.portugalmail.net/servicos/email/portal-email-asp, acedido a última vez em 28 de Janeiro de 2010.

[PC10f] S.A. Portugalmail Comunicações. Portugalmail, 2010. Disponível em http://www.portugalmail.net/, acedido a última vez em 28 de Janeiro de 2010.

[PC10g] S.A. Portugalmail Comunicações. Portugalmail :: Quem somos, 2010. Disponível em http://www.portugalmail.pt/quemSomos/, acedido a última vez em 28 de Janeiro de 2010.

[Per09] Sarah Perez. Facebook eats away at email usage on today's web, 2009. Publicado em 17 de Setembro de 2009, disponível em http://www.readwriteweb.com/archives/facebook_eats_away_at_email_usage_on_todays_web.php, acedido a última vez em 28 de Janeiro de 2010.

[Pro10a] ProcessOne. ejabberd - High-Performance enterprise instant messaging - ProcessOne, 2010. Disponível em http://www.process-one.net/en/ejabberd/desc, acedido a última vez em 28 de Janeiro de 2010.

[Pro10b] ProcessOne. tweet.IM - control your twitter with your instant messenger - powered by ProcessOne. https://www.tweet.im/content, 2010. Disponível em http://xmpp.org/extensions/xep-0077.html, acedido a última vez em 22 de Junho de 2010.

[PSSM09] Ian Paterson, Dave Smith, Peter Saint-Andre e Jack Moffitt. XEP-0124: bidirectional-streams over synchronous HTTP (BOSH), 2009. Disponível em http://xmpp.org/extensions/xep-0124.html, acedido a última vez em 20 de Junho de 2010.

[RRGC06] António D. Reis, José F. Rocha, Atilio S. Gameiro e José P. Carvalho. Sistemas de comunicação síncrona e assíncrona de dados. 2006. Publicado em 4 de Setembro de 2006 para a 15ª Conferência Nacional de Física, disponível em http://www2.fis.ua.pt/fisica2006_CD/pdf/FAP28%20-%20SISTEMAS%20DE%20COMUNICA%C3%87%C3%83O%20S%C3%8DNCRONA%20E%20ASS%C3%8DNCRONA%20DE.pdf, acedido a última vez em 28 de Janeiro de 2010.

[Sai04] Peter Saint-Andre. Extensible messaging and presence protocol (XMPP): instant messaging and presence, 2004. Publicado em Outubro de 2004, disponível em http://xmpp.org/rfcs/rfc3921.html, acedido a última vez em 15 de Junho de 2010.

[Sai09a] Peter Saint-Andre. [jdev] plaintext passwords hack, 2009. Publicado em 17 de Dezembro de 2009, disponível em http://mail.jabber.org/pipermail/jdev/2009-December/087950.html, acedido a última vez em 28 de Janeiro de 2010.

[Sai09b] Peter Saint-Andre. XEP-0077: In-Band registration, 2009. Actualizado em 15 de Setembro de 2009, disponível em http://xmpp.org/extensions/xep-0077.html, acedido a última vez em 3 de Junho de 2010.

[Sch09] Andy Schroepfer. Is email dead? | rackspace email & apps blog, 2009. Publicado em 18 de Setembro de 2009, disponível em http://www.rackspace.com/email_hosting/blog/2009/09/is-email-dead/, acedido a última vez em 28 de Janeiro de 2010.

[Sie09] George Siemens. A fundamental shift in how we communicate: George siemens, twitter and the Real-Time web. 2009. Publicado em 25 de Setembro de 2009 para o MasterNewMedia, disponível em http://www.masternewmedia.org/a-fundamental-shift-in-how-we-communicate-george-siemens, acedido a última vez em 28 de Janeiro de 2010.

[SM05] Peter Saint-Andre e Ralph Meijer. Streaming XML with Jabber/XMPP. 2005. Publicado em 1 de Maio de 2005.

[Sof10] Jive Software. Openfire scalability. 2010. Disponível em http://www.igniterealtime.org/about/OpenfireScalability.pdf, acedido a última vez em 21 de Janeiro de 2010.

[Tig09a] Tigase.org. 1mln or more online users on the tigase cluster | tigase.org, 2009. Publicado em 26 de Novembro de 2009, disponível em http://www.tigase.org/content/1mln-or-more-online-users-tigase-cluster, acedido a última vez em 28 de Janeiro de 2010.

[Tig09b] Tigase.org. Tigase load tests again - 500k user connections | tigase.org, 2009. Publicado em 26 de Novembro de 2009, disponível em http://www.tigase.org/content/tigase-load-tests-again-500k-user-connections, acedido a última vez em 16 de Maio de 2010.

[Vas10] Jessica E. Vascellaro. The end of the email era - WSJ.com. 2010. Disponível em http://online.wsj.com/article/SB10001424052970203803904574431151489408372.html, acedido a última vez em 28 de Janeiro de 2010.

[Wig09] Adam Wiggins. Push systems vs pull systems, 2009. Publicado em 16 de Março de 2009, disponível em http://adam.heroku.com/past/2009/3/16/push_systems_vs_pull_systems/, acedido a última vez em 25 de Junho de 2010.

Anexo A

Diagrama da Arquitectura do Raindrop

Neste anexo encontra-se uma breve descrição e o diagrama de arquitectura do Mozilla Raindrop.

A.1 Arquitectura do Raindrop

Como se pode constatar na Figura A.1[1] o *back end* da aplicação é um servidor interno do CouchDB. Este comunica com os componentes responsáveis por processar e recuperar mensagens que estão codificados em Python dando uso à Framework Twisted Networking. O CouchDB é depois responsável de enviar essas mensagens para a interface.

[1] Fonte: https://wiki.mozilla.org/Raindrop/SoftwareArchitecture

Diagrama da Arquitectura do Raindrop

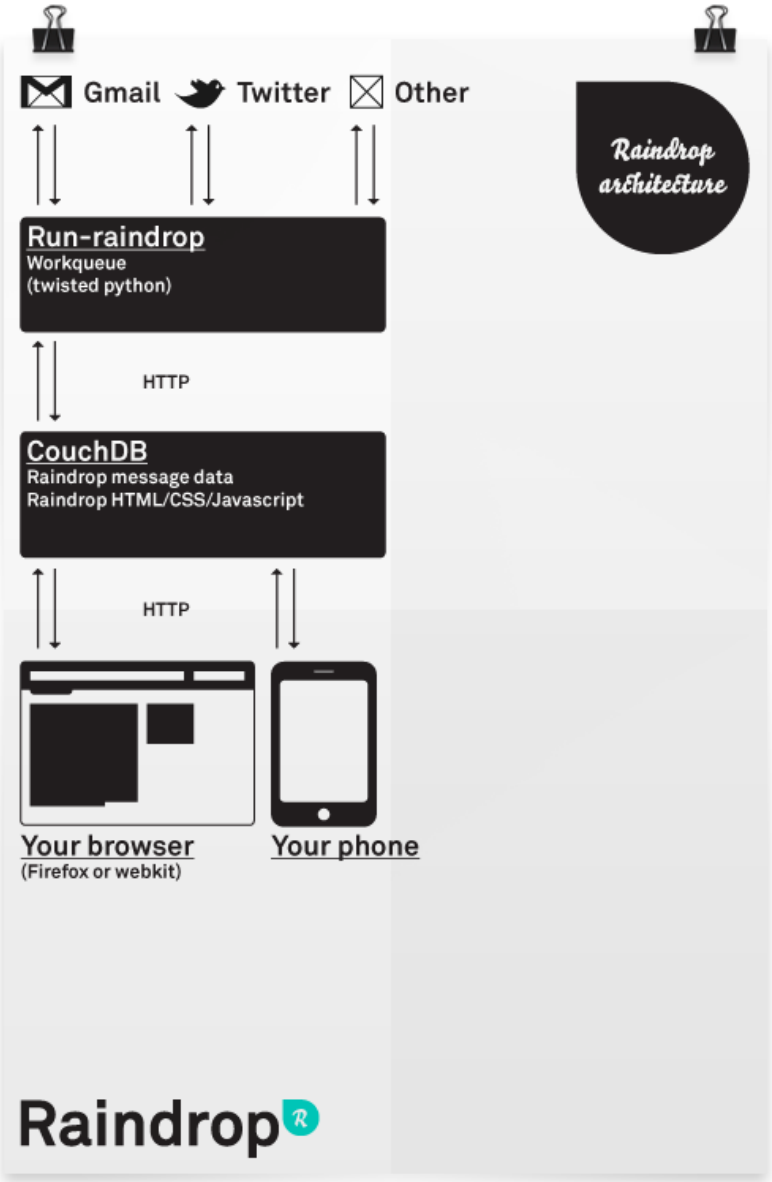


Figura A.1: Arquitectura do Raindrop.

Anexo B

Diagramas do XMPP

Neste anexo encontram-se diagramas de arquitectura e funcionamento do protocolo XMPP.

B.1 Diagramas de Arquitectura do XMPP

Nas figuras é apresentada a arquitectura do protocolo XMPP. A Figura B.1 é uma visão simplificada da comunicação entre clientes em servidores diferentes. A Figura B.2 é uma visão mais alargada dos vários componentes que podem pertencer a um sistema que use o protocolo XMPP e a maneira como se efectuam as ligações entre estes.

B.2 Diagrama de Funcionamento do XMPP

A Figura B.3[1] ilustra uma possibilidade de construção das *streams* XMPP. Exemplificando a forma como o documento XML é construído ao longo da comunicação.

[1] Fonte: http://www.ibm.com/developerworks/library/x-xmppintro/

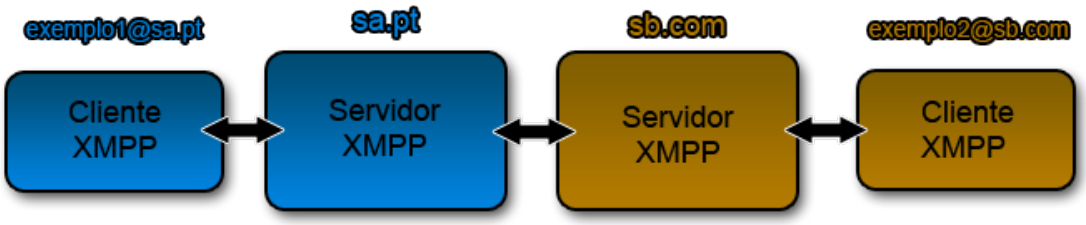


Figura B.1: Diagrama do Cliente/Servidor XMPP.

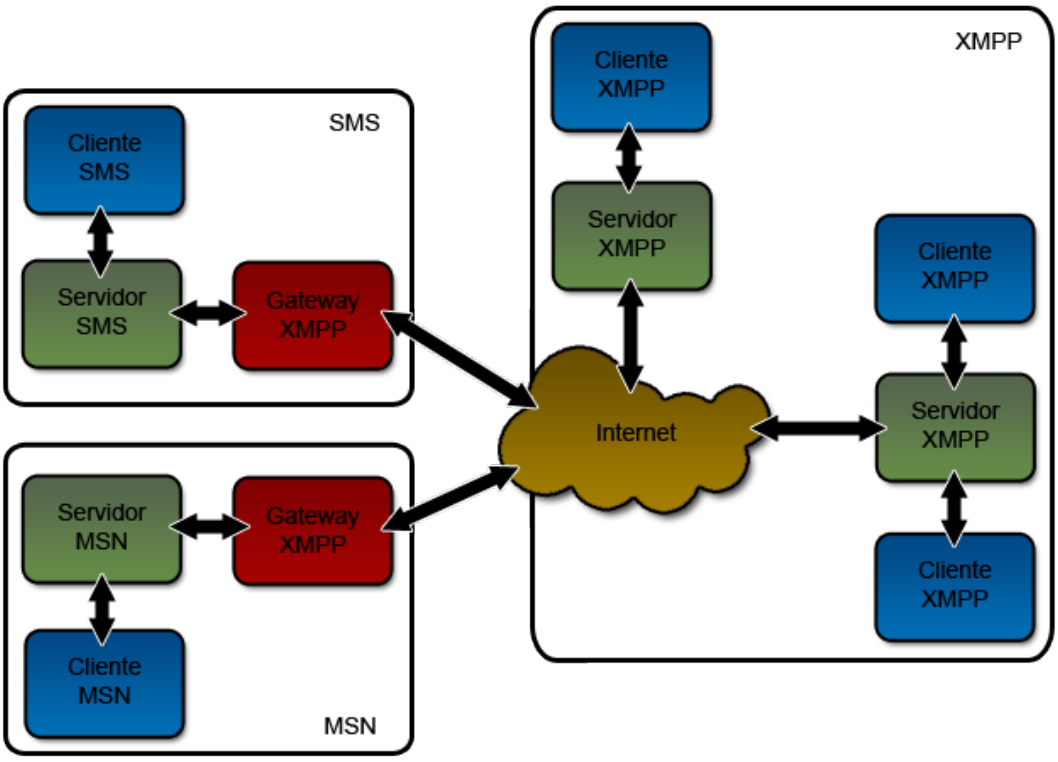


Figura B.2: Diagrama da arquitectura incluindo transportes.

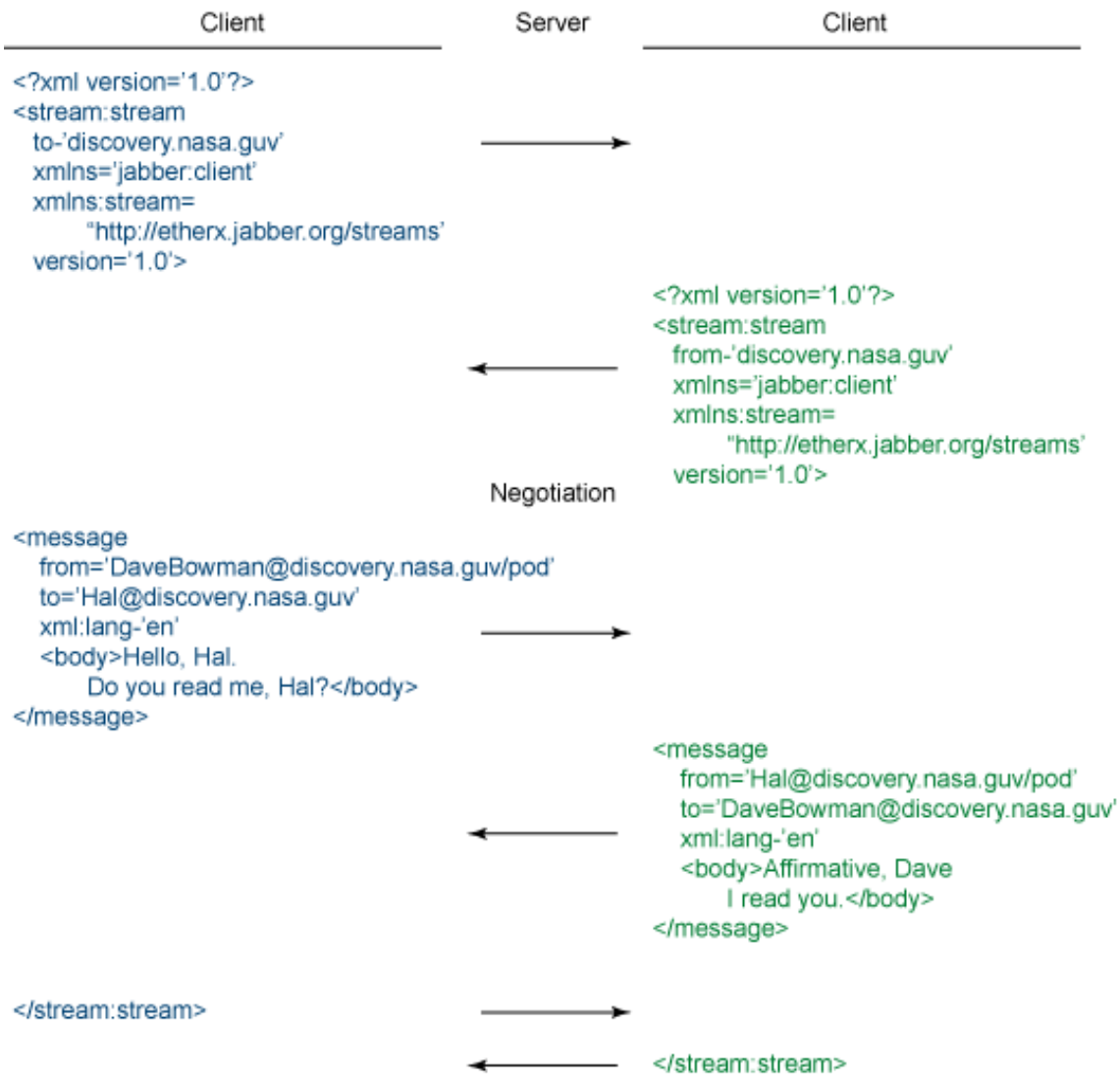


Figura B.3: Diagrama de comunicação XMPP.

Anexo C

Benchmark aos Servidores XMPP

Neste apêndice encontra-se informação relativa ao *benchmark* efectuado aos servidores XMPP Ejabberd, Openfire e Tigase Server com a ferramenta Tsung.

C.1 Teste Efectuado

Os testes no Tsung são definidos no formato XML. Nesta secção encontra-se o teste e uma explicação do seu conteúdo.

O teste divide-se em duas partes: na primeira encontram-se as *tags* de configuração do teste, na segunda encontra-se a *tag sessions* onde são definidas as acções que cada utilizador criado irá efectuar.

Das configurações a parte que interessa destacar é a *arrivalphase*. Nessa *tag* é definida a duração do teste, 10 m, e a frequência de chegada de utilizadores, um utilizador é criado a cada 0,07 s.

As transacções realizadas por cada utilizador,com tempos de espera entre estas definidas nas *tags thinktime*, são as seguintes:

- autentica-se (no XML identificada como *authenticate*)
- envia a *stanza* de presença inicial
- requisita a lista de contactos (no XML identificada como *rosterget*)
- envia a *stanza* de presença que o identifica como disponível para trocar mensagens com outros utilizadores (no XML identificada como *online*)
- envia a *stanza* de presença que o identifica como indisponível para trocar mensagens com outros utilizadores (no XML identificada como *offline*)
- actualiza o estado para *Hello*
- adiciona um utilizador e subscreve a sua presença (no XML identificada como *rosteradd*)
- envia novamente a *stanza* de presença que o identifica como disponível para trocar mensagens com outros utilizadores(no XML identificada como *keeplive*)
- envia a *stanza* de presença final

- acaba a ligação (no XML identificada como *close*)

```
<?xml version="1.0" encoding="UTF-8"?>
<tsung loglevel="notice" version="1.0">

  <clients>
    <client host="testclient" use_controller_vm="true" maxusers="10000"/>
  </clients>

  <servers>
  <server host="testserver" port="5222" type="tcp"/>
 </servers>

 <monitoring>
    <monitor host="testserver" type="munin"/>
    <monitor host="testclient" type="munin"/>
  </monitoring>

 <load>
  <arrivalphase phase="1" duration="10" unit="minute">
   <users interarrival="0.07" unit="second"/>
  </arrivalphase>
 </load>

  <options>
  <option type="ts_jabber" name="global_number" value="20001"/>
  <option type="ts_jabber" name="userid_max" value="20000"/>
  <option type="ts_jabber" name="domain" value="testserver"/>
  <option type="ts_jabber" name="username" value="user"/>
  <option type="ts_jabber" name="passwd" value="pass"/>
 </options>

  <sessions>
   <session bidi="true" probability="100" name="jabber-mytest" type="ts_jabber"
       >

    <request> <jabber type="connect" ack="no_ack"/> </request>
    <thinktime value="1"/>
    <transaction name="authenticate">
      <request> <jabber type="auth_get" ack="local"/> </request>
      <request> <jabber type="auth_set_plain" ack="local"/> </request>
    </transaction>

    <request> <jabber type="presence:initial" ack="no_ack"/> </request>
    <thinktime value="1"/>

    <transaction name="rosterget">
      <request> <jabber type="iq:roster:get" ack="local"/></request>
    </transaction>

    <thinktime value="3"/>

    <transaction name="online">
    <request> <jabber type="chat" ack="no_ack" size="16" destination="online"/>
        </request>
    </transaction>

    <thinktime value="3"/>

    <transaction name="offline">
```

```
57        <request> <jabber type="chat" ack="no_ack" size="56" destination="offline
              "/> </request>
58      </transaction>
59
60      <thinktime value="12"/>
61
62      <request> <jabber type="presence:broadcast" show="chat" status="Hello" ack=
            "no_ack"/> </request>
63
64     <thinktime value="6"/>
65
66      <transaction name="rosteradd">
67        <request> <jabber type="iq:roster:add" ack="no_ack" destination="online"/
              > </request>
68        <request> <jabber type="presence:subscribe" ack="no_ack"/> </request>
69      </transaction>
70
71      <thinktime value="30"/>
72
73      <transaction name="keepalive">
74       <request> <jabber type="chat" ack="no_ack" size="1" destination="online"/>
              </request>
75      </transaction>
76
77      <thinktime value="3"/>
78
79      <request> <jabber type="presence:final" ack="no_ack"/> </request>
80      <thinktime value="1"/>
81
82      <transaction name="close">
83        <request> <jabber type="close" ack="no_ack"/> </request>
84      </transaction>
85
86    </session>
87
88   </sessions>
89 </tsung>
```

C.2 Resultados

Nas Tabelas C.1, C.3 e C.5 encontra-se informação sobre a taxa de ocorrências por segundo, a duração média e a quantidade total de ocorrências de 3 acções ou estados dos utilizadores: a conexão ao servidor, o pedidos efectuado ao servidor e a sessão de cada utilizador.

Nas Tabelas C.2, C.4 e C.6 encontra-se informação sobre a taxa de ocorrências por segundo, a duração média e a quantidade total de ocorrências das transacções definidas no teste da Secção C.1.

Nas Figuras C.1, C.5 e C.9 são apresentados gráficos de tráfego de rede. Demonstram a quantidade de dados enviados e recebidos pelo servidor durante o tempo do teste.

Nas Figuras C.2, C.6, C.10, C.3, C.7, C.11, C.4, C.8 e C.12 são apresentados gráficos da utilização do processador, memória livre e carga das máquinas que estavam a servir de cliente e servidor durante o teste, respectivamente instalação Tsung e servidor XMPP.

Tabela C.1: Estatísticas globais do Ejabberd.

Nome	Taxa	Duração média	Quantidade total
Conexão	16,3/s	0,921 ms	8370
Pedidos	49,4/s	15,420 ms	25110
Sessão	17,3/s	60 000 ms	8370

C.3 Conclusões

O Tigase Server de um modo geral foi o servidor que obteve os piores resultados. Este obteve resultados bastante fracos na autenticação e na obtenção da lista de contactos. Analisando os parâmetros do Openfire verifica-se que foi o servidor mais rápido a obter a lista de contactos, por outro lado foi o servidor que gerou menos carga sobre a máquina. O Ejabberd foi o servidor que se destacou mais obtendo os melhores valores em termos de carga e uso do processador o que significa que suportaria mais utilizadores e mais operações que qualquer um dos outros na mesma máquina, também se destacou na transacção de autenticação obtendo o valor mais rápido.

Tabela C.2: Estatísticas das transacções efectuadas no Ejabberd.

Nome	Taxa	Duração média	Quantidade total
authenticate	16,4/s	9,980 ms	8370
close	18,2/s	0,164 ms	8370
keepalive	16,8/s	0,221 ms	8370
offline	16,6/s	0,230 ms	8370
online	16,6/s	0,223 ms	8370
rosteradd	16,8/s	0,314 ms	8370
rosterget	16,6/s	36,690 ms	8370

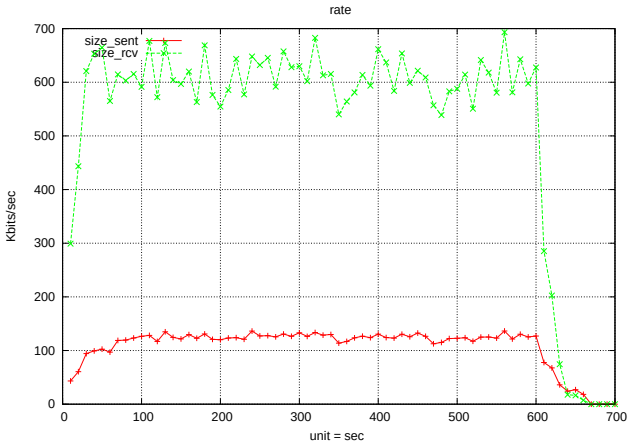


Figura C.1: Gráfico de tráfego de rede do Ejabberd.

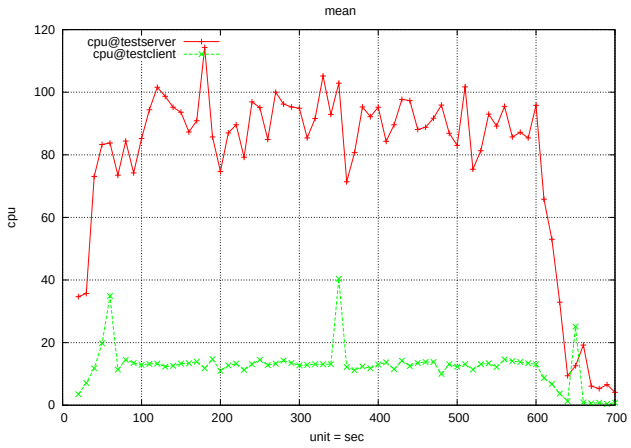


Figura C.2: Gráfico de utilização do processador na máquina do Tsung (cliente) e na máquina do Ejabberd (servidor).

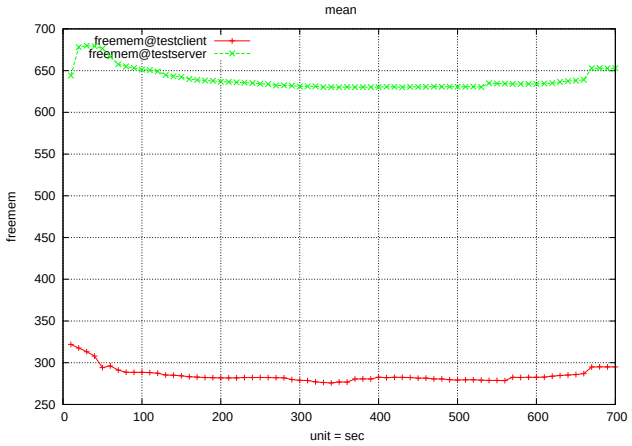


Figura C.3: Gráfico da memória livre na máquina do Tsung (cliente) e na máquina do Ejabberd (servidor).

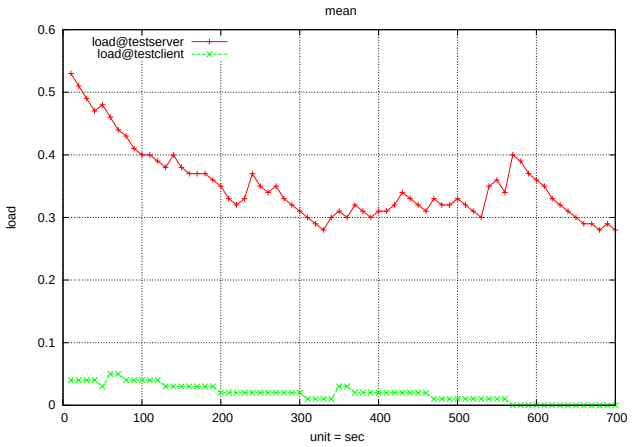


Figura C.4: Gráfico da carga na máquina do Tsung (cliente) e na máquina do Ejabberd (servidor).

Tabela C.3: Estatísticas globais do Openfire.

Nome	**Taxa**	**Duração média**	**Quantidade total**
Conexão	16,8/s	0,900 ms	8467
Pedidos	49,6/s	18,000 ms	25401
Sessão	16,4/s	60 000,000 ms	8467

Tabela C.4: Estatísticas das transacções efectuadas no Openfire.

Nome	Taxa	Duração média	Quantidade total
authenticate	16,7/s	31,940 ms	8467
close	16,5/s	0,166 ms	8467
keepalive	16,8/s	0,219 ms	8467
offline	17,0/s	0,221 ms	8467
online	16,7/s	0,219 ms	8467
rosteradd	17,0/s	0,351 ms	8467
rosterget	17,1/s	22,510 ms	8467

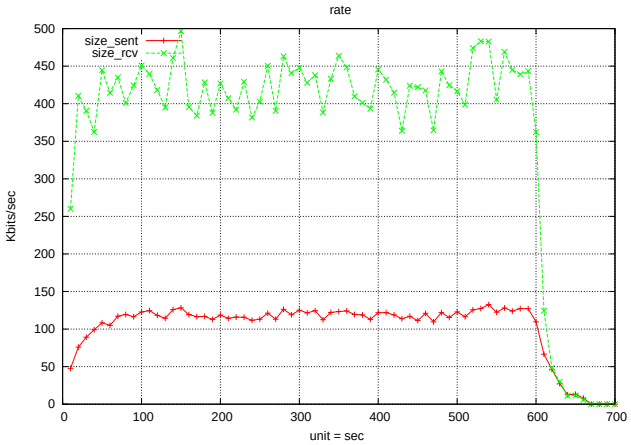


Figura C.5: Gráfico de tráfego de rede do Openfire.

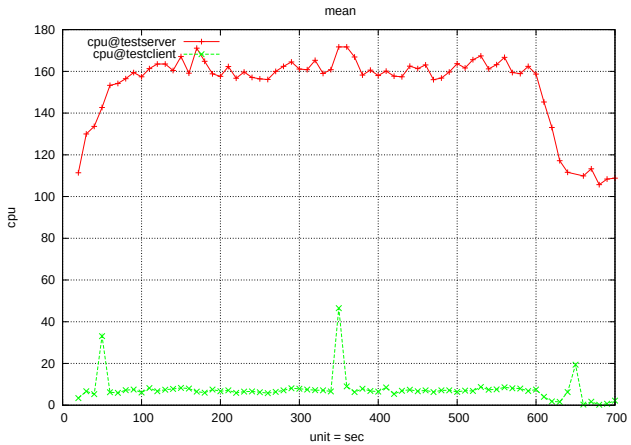


Figura C.6: Gráfico de utilização do processador na máquina do Tsung (cliente) e na máquina do Openfire (servidor).

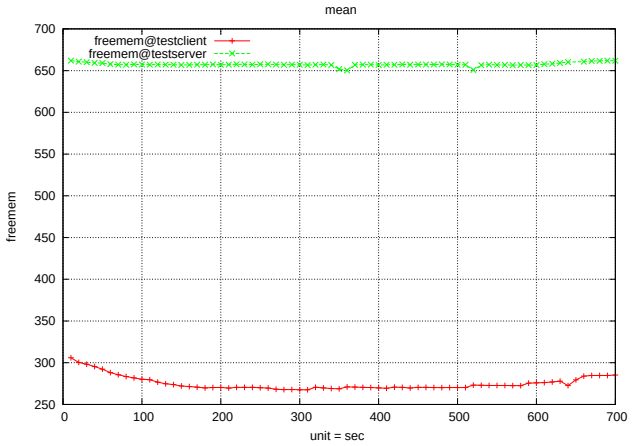


Figura C.7: Gráfico da memória livre na máquina do Tsung (cliente) e na máquina do Openfire (servidor).

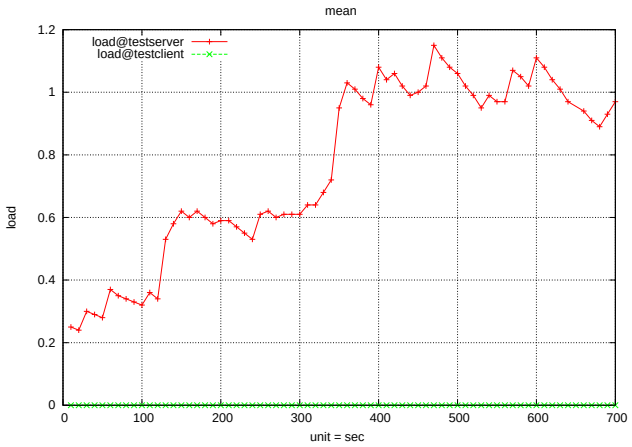


Figura C.8: Gráfico da carga na máquina do Tsung (cliente) e na máquina do Openfire (servidor).

Tabela C.5: Estatísticas globais do Tigase Server.

Nome	**Taxa**	**Duração média**	**Quantidade total**
Conexão	16,8/s	0,971 ms	8389
Pedidos	452,8/s	140,000 ms	25155
Sessão	17,2/s	60 000,000 ms	8385

Tabela C.6: Estatísticas das transacções efectuadas no Tigase Server.

Nome	Taxa	Duração média	Quantidade total
authenticate	17,5/s	300,000 ms	8385
close	17,5/s	0,169 ms	8385
keepalive	17,8/s	0,225 ms	8385
offline	17,3/s	0,231 ms	8385
online	17,2/s	0,221 ms	8385
rosteradd	17,8/s	0,349 ms	8385
rosterget	18,5/s	120,000 ms	8385

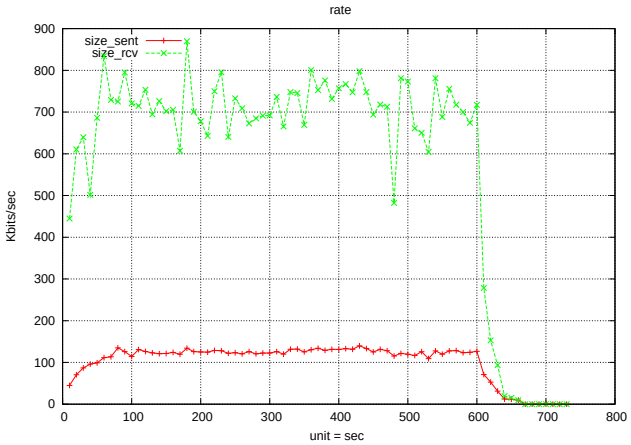


Figura C.9: Gráfico de tráfego de rede do Tigase Server.

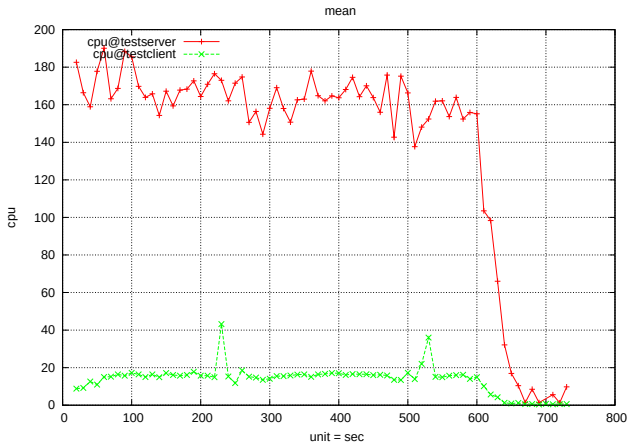


Figura C.10: Gráfico de utilização do processador na máquina do Tsung (cliente) e na máquina do Tigase Server (servidor).

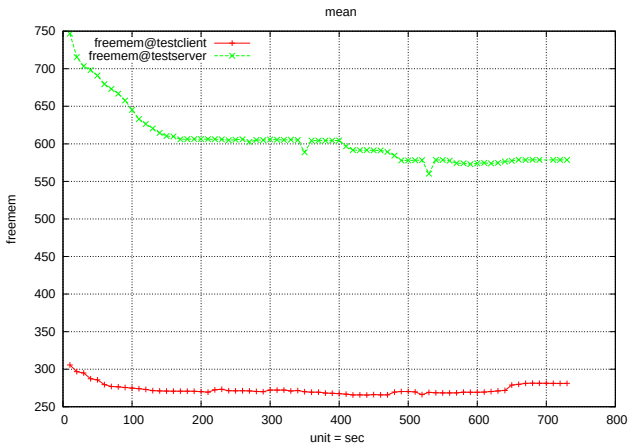


Figura C.11: Gráfico da memória livre na máquina do Tsung (cliente) e na máquina do Tigase Server (servidor).

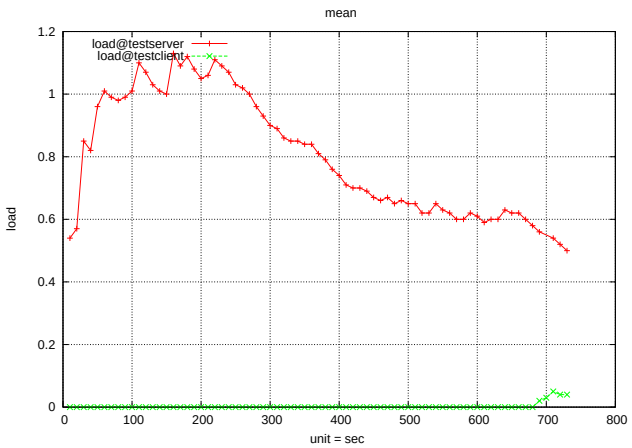


Figura C.12: Gráfico da carga na máquina do Tsung (cliente) e na máquina do Tigase Server (servidor).