# PUC – Sistema de Comunicações Pessoais para as Redes de Próxima Geração

Alberto Rodrigues Silva, João Patriarca, João Clemente, Paulo Chainho, Paulo Ferreira

Instituto Superior Técnico / INESC-ID
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
+351.21.3100307

alberto.silva@acm.org

## Resumo

*Introduz-se neste artigo o enquadramento e motivação para a realização do projecto "PUC – Sistema de Comunicações Pessoais para as Redes de Próxima Geração". O artigo tem como principal objectivo apresentar a visão geral do projecto PUC, focando-se particularmente no serviço "myComs – Serviço Pessoal de Comunicações", descrevendo os seus requisitos funcionais e não funcionais, assim como os aspectos relevantes relativamente ao desenho e implementação, e às arquitecturas tecnológica e de software de suporte.*

**Palavras-Chave**: 3GN, J2EE, Clustering, Email, IM, Jabber

## 1. Introdução

Na sequência dos avanços na área das tecnologias de informação e comunicação, de grandes expectativas que foram criadas no período especulativo das "dot coms", e consequentemente dos enormes investimentos financeiros que têm sido realizado pelas operadoras de telecomunicações, existe actualmente uma necessidade imperiosa de se criarem novos serviços que sustentem e rentabilizem os investimentos realizados.

As operadoras de telecomunicações têm vindo progressivamente, e em todo o mundo, a substituir as suas redes antigas (analógicas e ou digitais, orientadas fundamentalmente por serviços de voz baseados na telefonia) por redes de nova geração, onde se estima que os serviços de dados e multimédia ultrapasse, em termos de tráfego da rede, os tradicionais serviços de voz. Estas redes integram e disponibilizam um conjunto de características técnicas extremamente avançadas, integrando serviços de comunicação fixo com móvel, voz com dados simples (e.g., SMS) ou multimédia (e.g., MMS), e ainda providenciando acesso a outras redes externas (e.g., Internet, iTV).

No actual contexto de depressão económica, em que as operadoras de telecomunicações efectuaram enormes investimentos, é crítico a concepção e desenvolvimento de novos serviços que possam sustentar e potenciar os investimentos realizados; é neste âmbito que se enquadra o trabalho descrito neste artigo.

O "**PUC – Sistema de Comunicações Pessoais para as Redes de Próxima Geração**" [1] é um projecto de I&D, resultado de uma colaboração entre o INESC-ID e a PT-Inovação, tendo como principal objectivo a concepção de um sistema constituído por múltiplos serviços pessoais que tirem partido do estado emergente das tecnologias telemáticas e das redes de telecomunicações de próxima geração.

O sistema PUC pretende oferecer aos seus subscritores um conjunto alargado de serviços pessoais, de entre os quais se destacam (1) o serviço de comunicações pessoais (*myComs*); (2) o serviço de contactos (*myContacts*); e (3) o serviço de agenda e de compromissos (*myAgenda*). Estes serviços por si só não são factor de inovação. O principal factor de inovação do projecto PUC reside no facto destes serviços ser **desenvolvidos e subscritos de forma dinâmica e integrada**, serem **suportados pelas redes de próxima geração**, serem **acedidos através de diferentes tipos de terminais** (e.g., telefone fixo ou móvel, PC ou PDA) e consequentemente oferecerem **diferentes tipos de interacção homem-máquina** (e.g., via voz, Web, Wap).

O artigo tem como principal objectivo apresentar a visão geral do projecto PUC, focando-se no serviço *myComs*, descrevendo os seus requisitos funcionais e não funcionais, assim como discutir os aspectos relevantes relativamente à arquitectura tecnológica e à arquitectura de software de suporte. O artigo encontra-se estruturado em 6 secções. A Secção 1 introduz o contexto e motivação do projecto PUC. A Secção 2 introduz os princípios orientadores e a visão geral do sistema PUC. A Secção 4 apresenta e discute tecnologias e produtos *open source* de terceiras partes usados no âmbito do projecto. As Secções 4 e 5 discutem detalhes de desenho, implementação e instalação relativamente aos requisitos funcionais e não funcionais do serviço *myComs*. Por fim, a Secção 6 apresenta o estado do projecto relativamente ao trabalho desenvolvido, em curso e para o futuro; e apresenta ainda as conclusões finais.

## 2. Visão e Objectivos Gerais

O sistema PUC tem como objectivo oferecer um conjunto alargado de serviços pessoais sobre uma infra-estrutura de rede de telecomunicações de próxima geração.

### 2.1 Princípios PUC

A concepção e o desenvolvimento do sistema PUC é orientada por um conjunto restrito de princípios, dos quais se destacam os seguintes.

**Abstracção das tecnologias de rede**

O PUC procura seguir os princípios conceptuais das redes de próxima geração, em particular a utilização de interfaces abertas de programação que abstraiam o programador das tecnologias de rede subjacentes. Em termos de arquitectura funcional estas interfaces são oferecidas por elementos de rede a que se designa "Servidor de Aplicações de Telecomunicações".

**Ubiquidade de tipo de acesso e de tipo de media**

O sistema deve poder ser acedido através de distintos tipos de terminais (e.g., telefone, telefone inteligente, PDA, PC) e segundo diferentes tipos de interacção homem-máquina (e.g., interacções baseadas em interface voz, web, wap, menus e écrans de telemóveis). Note-se no entanto, que nem todos os casos de utilização serão suportados integralmente ao nível dos vários tipos de terminais e de acessos.

**Elevado número de utilizadores e de transações**

O sistema deve suportar a subscrição e utilização de milhares ou milhões de utilizadores. Ou seja, o sistema deverá ser escalável, em termos do número de utilizadores e de transações, mantendo tempos de resposta adequados aos parâmetros de qualidade exigidos. Adicionalmente, tendo em conta a dimensão de utilizadores envolvida, o sistema deverá apresentar disponibilidade e tolerância a faltas próxima de 100%.

**Gestão e subscrição flexível e dinâmica de serviços**

O sistema deve permitir a gestão (e.g., introdução, suspensão, eliminação) de serviços pessoais de forma fácil e dinâmica. Deve ainda suportar a gestão de versões de forma a garantir e controlar a evolução graciosa dos serviços, i.e., sem forçar paragens do sistema.

Por outro lado, o sistema deve permitir aos seus utilizadores a gestão flexível e também dinâmica da subscrição dos serviços disponíveis. Neste âmbito deve ser suportado o conceito de "conta de utilizador global" no sistema (de forma que o utilizador não tenha de guardar e manter várias *passwords* conforme o número de serviços subscritos) bem como "conta de cliente global" (de forma que o sistema de *billing* emita apenas uma única factura independentemente dos serviços subscritos).

**Tecnologia Java e *Open-Source***

O sistema deve ser desenvolvido em Java, sobre as suas várias tecnologias emergentes, particularmente agregadas às especificações J2EE [2] e J2ME [3]. Adicionalmente, de forma a minimizar a "reinvenção da roda" tanto quanto possível e a acelerar o desenvolvimento do projecto, poderão ser usados e integrados projectos Java, desde que desenvolvidos por terceiras partes segundo a filosofia do software livre (i.e., *open-source*) [5,6].

### 2.2 Visão Geral

A Figura 10 ilustra a visão geral do sistema PUC. Este é constituído conceptualmente por um nó central, designado por "Servidor Aplicacional PUC" (que integra diferentes servidores de recursos incluindo Servidor LDAP, Servidor Email, Servidor Jabber, Servidor Media VoiceXML) e por vários nós, de diferentes tipos, que desempenham o papel de terminal de acesso. Entre outros, podemos identificar os seguintes tipos de terminais e tipos de interface homem-máquina: PC com interface Web; telemóveis com interface Wap ou interface Windows (e.g., baseado no J2ME); ou telefones inteligentes.

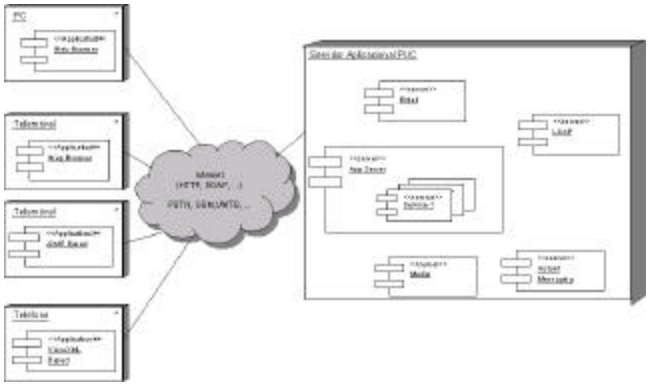

*Figura 1: Visão geral da arquitectura do sistema PUC.*

O nó "Servidor PUC" executa os componentes de software fundamentais com perfil servidor, designadamente os componentes correspondentes aos serviços disponibilizados (e.g., Serviço-1, Serviço-2, ... Serviço-N). Note-se que o "Servidor PUC" representado apenas por um nó, por motivos de simplicidade, é na realidade concretizado por vários nós distribuídos de modo a garantir a escalabilidade e desempenho do sistema correspondente ao princípio de suporte de um "elevado número de utilizadores e de transações".

Por outro lado, de forma a satisfazer o princípio da "ubiquidade de tipo de acesso e de tipo de media", deve permitir-se o acesso ao sistema a partir de vários terminais, os quais podem executar componentes pré-instaladas e específicas do sistema PUC (e.g., componente J2ME instalada nos telemóveis) ou apenas componentes gerais e não específicas (e.g.,

um Web/Wap-browser instalado num PC ou num telemóvel).

### 2.3 Principais Serviços

Tendo em conta o princípio de "gestão e subscrição flexível e dinâmica de serviços" o sistema deverá suportar com flexibilidade quer a gestão de serviços (i.e. registo de novo serviço, suspensão, evolução para nova versão, remoção) quer a sua subscrição.

A operação de *gestão de serviços*, sendo crítica, deverá ser realizada apenas e exclusivamente pelos administradores e gestores do PUC, i.e., por empregados da empresa que opere o sistema.

Por outro lado, a *subscrição de serviços* é realizada pelos utilizadores-subscritores, na medida que estes poderão alterar de forma fácil e flexível os serviços que pretendem e posteriormente usá-los de forma integrada com todos os restantes já subscritos. Entre outros aspectos, deve ser suportado de forma independente dos serviços subscritos, os requisitos de "conta de utilizador (login e *password*) única" e de "factura (*billing*) única".

Na fase actual do projecto, foram definidos três tipos de serviços que se encontram em desenvolvimento, designadamente:

- **myComs – Serviço Pessoal de Comunicações**: Tem como objectivo oferecer aos seus utilizadores um leque alargado e integrado de recursos e serviços de comunicação, variando desde o serviço de correio electrónico (EMail), até ao de mensagens instantâneas e de presença (IM, *Instant Messaging*) passando pelo serviço de comunicação tradicional de telefonia, baseado em voz, ou pelo suporte de espaços Web pessoais e possibilidade de gestão de *bookmarks* e preferências.
- **myContacts – Serviço Pessoal de Contactos**: Tem como objectivo permitir aos seus utilizadores a gestão de contactos, os quais podem corresponder a contactos individuais e ou institucionais/empresariais. Existirão mecanismos avançados de referência dinâmica a contactos públicos mantidos por outros utilizadores ou pela própria empresa operadora do PUC. Desta forma, podem-se estabelecer ligações virtuais a contactos mantidos por entidades terceiras, com a garantia de que quando a informação correspondente é alterada (e.g., alteração de um número de telefone) todos os utilizadores com essas referências virtuais têm acesso à informação mais actualizada. Adicionalmente, existirão mecanismos de sincronização entre os contactos mantidos centralmente no servidor PUC e os mantidos localmente num telemóvel pessoal.
- **myAgenda – Serviço Pessoal de Agenda**: Tem como objectivo permitir aos seus utilizadores a gestão de agenda, com as operações clássicas de marcação de reuniões, compromissos, eventos, etc. De forma a suportar a realização de algumas operações de forma colaborativa (e.g., marcação de reunião), o dono de cada agenda poderá conceder privilégios de acesso e mesmo alteração da sua agenda a outros utilizadores definidos em grupos especiais para o efeito. (Por exemplo, poderá permitir que os utilizadores definidos no seu grupo "Secretárias" possam ter acesso à sua agenda e fazer quaisquer marcações no período livre dos dias úteis, entre as 9h e as 17h.) À semelhança com o serviço myContacts, existirão mecanismos de sincronização entre a agenda mantida centralmente no servidor PUC e a mantida localmente num telemóvel pessoal.

Por motivos de objectividade e tendo em conta o próprio estado actual do projecto, a apresentação das próximas secções do artigo será focada principalmente sobre o serviço *myComs*.

## 3. Tecnologias de Suporte

Nesta secção são apresentadas alguns aspectos relativos a tecnologias e ou produtos *open source* usadas no desenvolvimento do projecto PUC.

### 3.1 Java™ 2 Platform Enterprise Edition

O J2EE [2] é a especificação da SUN para plataformas de suporte de aplicações empresariais desenvolvidas em Java. A plataforma J2EE é baseada num modelo de aplicações distribuídas multicamada oferecendo entre outras as seguintes características: (1) independência do sistema operativo e da arquitectura computacional; (2) a possibilidade de reutilização de componentes de software; (3) suporte transparente à persistência e ao processamento transaccional; e (4) modelo unificado de segurança.

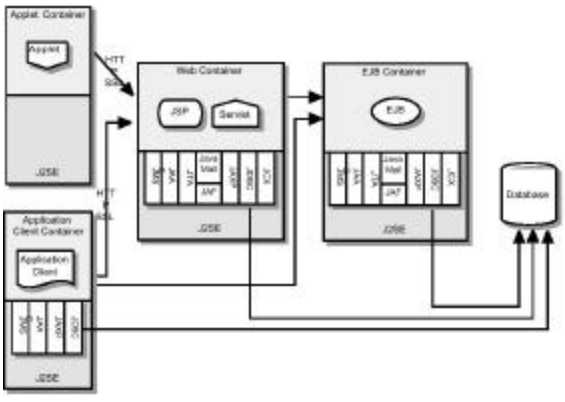


*Figura 2: Arquitectura simplificada do J2EE.*

A Figura 2 ilustra os principais módulos de uma aplicação desenhada (e desenvolvida) segundo a arquitectura J2EE. Em particular são evidentes a existência de distintos domínios de execução, designados geralmente por "contentores". Em geral, uma aplicação empresarial J2EE encontra-se

distribuída por vários componentes executados nos seguintes domínios: contentor de applets (tipicamente *applets* executados no contexto de Web *browsers*); contentor Web (tipicamente *servlets e JSP*); *e contentor* EJB (tipicamente EJB, *enterprise Java beans*, também designados por "componentes empresariais") [10].

### 3.2 Jboss

O Jboss [4] é um servidor aplicacional J2EE desenvolvido e promovido pelo JBoss Group de acordo com o paradigma do *open source*, sendo distribuído segundo licença GLGPL (*GNU Lesser General Public License*). O Jboss é um servidor muito popular e usado em inúmeras soluções comerciais.

O Jboss é constituído por diferentes componentes incluindo nomeadamente: JbossServer, o contentor básico de EJB; JbossMQ para as mensagens JMS; JBossMX para email; JbossTX para transações JTA/JTS; JbossSX para segurança baseada em JAAS; JbossCX para conectividade JCA; e JBossCMP para gestão de persistência CMP persistence. O Jboss possibilita a integração modular de todos estes componentes através da arquitectura JMX (*Java Management eXtension*) que permite substituir qualquer dos referidos componente por um outro, desde que seja compatível com a arquitectura JMX e implemente as mesmas API. (Tal modularidade possibilita o uso de outras componentes que não sejam fornecidas necessariamente pela própria empresa JBoss Group.)

Uma das características mais apreciadas no Jboss é o designado *hot deployment*: a instalação de um componente de aplicação é tão simples como copiar pacotes JAR para o directório de instalação. Se esta operação for efectuada quando uma versão anterior da aplicação estiver carregada, a nova versão substitui a mais antiga sem conflitos e sem ser necessário parar o servidor, i.e., não terá qualquer impacto noutras aplicações que estejam a correr no mesmo servidor.

## 3.3 JWMA

O JWMA (*Java Web Mail Architecture*) [7] é um projecto Java *open-source* que oferece uma solução de correio electrónico com interface Web.

O JWMA suporta-se nomeadamente nas seguintes tecnologias/API: *Java Mail API*, para acesso ao servidor de email; e *Java Server Pages* e *Java Servlet API*, para gestão e produção de interfaces homem-máquina Web.

A Figura 3 ilustra a arquitectura do *JWMA*, a qual segue o padrão MVC (*Model-View-Control*) [17,18], padrão arquitectural bastante conhecido que distingue claramente componentes de modelo, de componentes de controlo, e de componentes da apresentação.

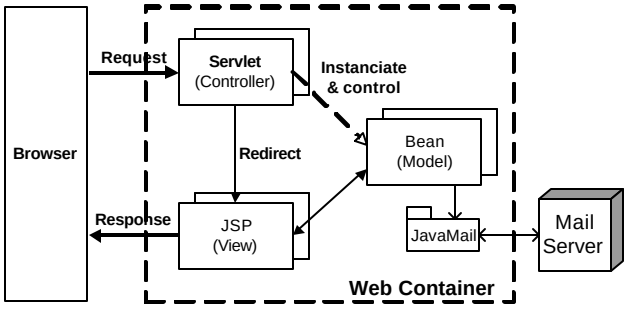


*Figura 3: Arquitectura de software do JWMA.*

Neste caso concreto os *servlets* desempenham o papel de componentes de controlo; são o ponto de entrada da aplicação, ou seja, recebem os pedidos efectuados pelo utilizador. Os *beans* (Java Beans) desempenham o papel de componentes do modelo; é através deles que é realizado o acesso ao servidor de email. As *JSP* (*Java Server Pages*) desempenham o papel de componentes de apresentação; através delas são criadas as páginas HTML com que o utilizador vai interagir.

### 3.4 Jabber

O Jabber é um protocolo de comunicação, definido em XML, que disponibiliza um conjunto de funcionalidades muito potentes e extensíveis [8]. Recentemente, foi aprovada a criação de um grupo de trabalho do IETF para a normalização deste protocolo definido, neste contexto, como XMPP (*eXtensible Messaging and Presence Protocol*) [22].

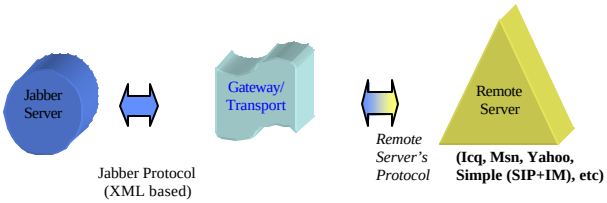


*Figura 4: Arquitectura Jabber.*

O Jabber é um protocolo de "Presença e de Mensagens Instantâneas" definido e desenvolvido de acordo com o paradigma do o*pen source*. Esta tecnologia é suportada por uma grande comunidade de utilizadores e de programadores. Apesar de ser uma tecnologia o*pen source* o Jabber tem um forte apoio da indústria. sendo o Jabber.com o seu principal representante comercial (e.g., o serviço "Mensageiro" do Portal SAPO [21] é fornecido através esta organização).

O protocolo Jabber permite efectuar operações de rede adicionais ao nível da aplicação, e.g., pesquisas de bases de dados (perfil de utilizadores, serviços de directório), gestão de acesso, etc.

### 3.5 VoiceXML

O VoiceXML [23] é uma linguagem baseada em XML usada para criar interfaces homem-máquina baseadas em voz, especialmente usadas em terminais do tipo telefone. Usa reconhecimento de voz assim como as teclas do telefone (DTMF) como *input*; e áudio previamente gravado e síntese de texto para voz (*text-to-speech* :TTS) como *output*.

O VoiceXML foi desenhada para permitir um controlo total na definição de interacções ou de diálogos (falados) entre os utilizadores e as aplicações. O VoiceXML oferece várias capacidades, nomeadamente formas de controlar o *output* áudio, *input* áudio, lógica de apresentação e fluxo de controlo, manuseamento de eventos e ligações telefónicas básicas.

O VoiceXML facilita a rápida criação de novas aplicações e esconde dos programadores alguns detalhes de implementação. Por exemplo, separa a camada da interacção com o utilizador da camada lógica do serviço.

Uma aplicação VoiceXML consiste genericamente

- Num *Servidor de Media VoiceXML*, que corre um *interpretador* VoiceXML, o qual actua como cliente para o servidor de aplicações. Este interpretador compreende diálogos VoiceXML e controla recursos de voz e de telefonia. Estes recursos incluem ASR, TTS, funções de toque e gravação, assim como uma interface para a rede telefónica
- Num *Servidor de Aplicações* que gera o VoiceXML de acordo com a execução duma lógica da aplicação, que pode fazer uso duma base de dados ou servidor de transacções.

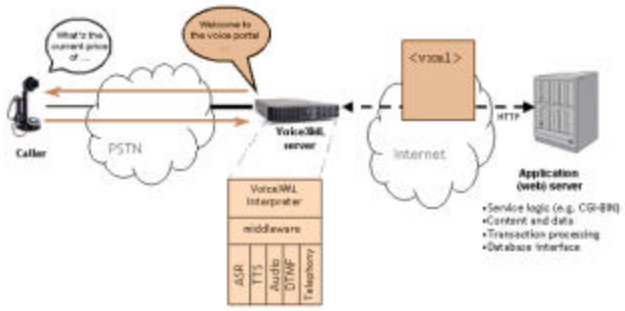


*Figura 5: Componentes duma aplicação VoiceXML.*

## 4. *myComs* – Requisitos Funcionais

Como referido na Secção 2.3 o *myComs* é um serviço pessoal de comunicações. Nesse sentido a ideia é integrar um leque variado de sub-serviços na área das telecomunicações, nomeadamente serviço de correio electrónico, serviço de mensagens instantâneas e notificações de presença, serviços relacionados com voz sobre IP, entre outros. Num cenário actual, para cada um destes serviços o utilizador necessita de instalar localmente uma aplicação para permitir o acesso ao respectivo serviço, precisa configurar essa aplicação, precisa registar-se perante o serviço, precisa manter contas para cada serviço, enfim... uma série de actividades que o utilizador precisa de realizar, quando o seu objectivo inicial era apenas aceder um determinado serviço. A integração de serviços vem permitir que o utilizador se autentique uma única vez para conseguir aceder a todos os serviços a que tem direito.

Outra característica do *myComs* é que os serviços são disponibilizados via Web. Esta particularidade permite minimizar qualquer tipo de intervenção no lado do utilizador. Em última instância o utilizador precisará apenas de um browser para aceder aos serviços. Tem igualmente a vantagem das actualizações dos serviços serem totalmente transparentes para o utilizador final.

### 4.1 Tipos de Utilizadores

O *myComs* é utilizado por diferentes utilizadores, cada qual associado a um determinado perfil de utilização, designados por actores. Estão previstos três tipos de actores: anónimo, membro e gestor.

O utilizador "anónimo" tem um acesso limitado, normalmente podendo consultar informação geral do sistema e/ou solicitar a subscrição do serviço. O utilizador "membro", após autenticação, tem um acesso completo às funcionalidades disponíveis. Neste âmbito pode ainda existir algum controle de acesso mediante o tipo de subscrição de serviços que fez. Por fim o utilizador "gestor" tem acesso completo à gestão de contas dos membros do sistema; em caso algum, o gestor poderá ter acesso directo às contas dos membros do *myComs* de modo a garantir a privacidade do serviço.

As próximas secções descrevem a arquitectura dos diferentes serviços implementados no âmbito do *myComs*, nomeadamente: o serviço de subscrição; o serviço de correio electrónico; e o serviço de mensagens instantâneas e notificações de presença.

### 4.2 Serviço Subscrição

Um utilizador para se tornar membro do *myComs*, e com isso usufruir de todos os serviços disponibilizados precisa de se registar no sistema, ou seja, de realizar a subscrição. Neste sentido a subscrição foi entendida como mais um serviço do *myComs*. Isso reflecte-se logo na arquitectura deste serviço que segue o modelo adoptado para todos os restantes. A subscrição, a nível funcional, não é mais do que recolher os dados do utilizador através de um conjunto de formulários, validar esses dados, e criar uma nova entrada no servidor de registos do utilizador (embora não implementado, terá também o objectivo de criar as demais contas necessárias nos

diferentes servidores para que os demais serviços possam ser disponibilizados).

### 4.3 Serviço Email

O serviço de email deverá permitir aos seus utilizadores várias funcionalidades, de entre as quais se destacam: a possibilidade de envio e recepção de mensagens de email; a gestão de caixas de correio (e.g., segundo o paradigma de pastas hierárquicas); e a gestão de contactos e endereços electrónicos. (Esta última funcionalidade sugere as fortes dependências entre os vários serviços do PUC, nomeadamente entre o *myComs* e o *myContacts*.)

De forma a integrar no *myComs* o serviço de email, foi adoptado e adaptado como projecto de base o sistema *open source* JWMA [7].

No entanto, conforme referido na Secção 3.3, o JWMA original baseia-se numa arquitectura Web simples, a duas camadas (servidor Web + servidor de email), i.e., o JWMA é executado apenas no contexto de um contentor Web, acedendo directamente ao servidor de email. Este facto levantou as subsequentes e inerentes dificuldades: (1) o software não pode ser reutilizado facilmente por outros componentes; (2) dificuldades em separar de facto a camada de apresentação da camada da lógica de negócio; (3) a nível de segurança; (4) dificuldades no suporte a transacções; e (5) no suporte à escalabilidade do serviço.

De forma a ultrapassar estas dificuldades foi proposta e implementada uma arquitectura adaptada (da versão original do JWMA) conforme sugerido na Figura 6.

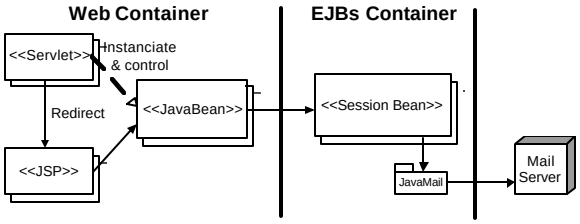


*Figura 6: Arquitectura final do serviço de email.*

A adaptação correspondeu a introduzir uma segunda camada entre o contentor Web e o servidor de email, designado por contentor de EJB. O acesso ao servidor de email passa a ser da responsabilidade dos componentes aí residentes, designados por EJB (*Enterprise Java Beans*) [10]. Para além dessa responsabilidade, os EJB representam o modelo de negócio. A utilização de EJB responde às dificuldades encontradas na versão original, nomeadamente: representando apenas o modelo, separam-no de facto da apresentação que pode ir ao nível físico, e como tal, facilmente para um mesmo modelo podem existir diferentes apresentações; os EJB disponibilizam interfaces remotas permitindo uma fácil utilização por outros componentes; é da responsabilidade do contentor de EJB dar suporte a transacções e suporte a nível de segurança libertando o programador dessa responsabilidade; para permitir, entre outras características, escalabilidade, o contentor de EJB dá suporte à criação de um ambiente distribuído com a criação de *clusters* que será assunto noutra secção adiante. A Figura 7 é um exemplo da adaptação que se fez ao nível dos *JavaBeans.* Está ilustrado apenas uma parte da arquitectura uma vez que para os restantes elementos o processo é idêntico.

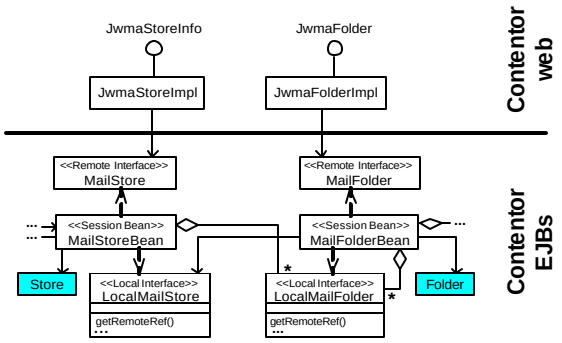


*Figura 7: Adaptação de JavaBeans a EJB.*

Em relação aos *JavaBeans* foram mantidas as interfaces, o que permitiu não ter que se alterar as classes que os usavam (*servlets* e *JSP*) mas ganharam referências remotas para os respectivos EJB. Cada EJB tem a si associado duas interfaces: interface remota, que permite o acesso por clientes remotos (e.g. *JavaBeans*); interface local, que é utilizada para manter localmente as relações entre EJB. Foram utilizadas interfaces locais por duas razões: primeiro, e admitindo que os vários EJB não têm sentido correr em contextos diferentes, ganha-se em eficiência usando estas interfaces; segundo, porque na interacção entre EJB são passados como parâmetros instâncias pertencentes à API do *JavaMail* [11]. Como estas classes não são serializáveis não podem ser passadas por valor, apenas por referência e como tal apenas num contexto local. De fazer notar ainda que a interface local disponibiliza um método que retorna uma interface remota em que ambas, a remota e a local, referenciam a mesma instância do *EJB*.

### 4.4 Serviço de Mensagens Instantâneas e de Presença

O serviço de mensagens instantâneas e presença é um serviço em tudo semelhante aos serviços proprietários conhecidos da Microsoft (*Messenger*) [12], *I Seek You* (*ICQ*) [13], da *AOL* (*AIM*) [14], entre outros.

Para implementação deste serviço foi utilizado uma biblioteca Java *open-source*, *JabberBeans* [9], catalogada também no portal *sourceforge* [6], que implementa o protocolo *Jabber* (conforme referido na Secção 3.4). Essa implementação é de relativo baixo nível já que apenas virtualiza a criação de

tramas em XML e a comunicação entre cliente servidor. Devido a este facto foi desenvolvido uma biblioteca Java, *xmpp*, que disponibiliza um conjunto de interfaces de mais alto nível. Nomeadamente, disponibiliza interfaces para autenticação do utilizador, para envio de mensagens e notificações de presença, bem como interfaces para notificação de mensagens e de presença por parte do servidor.

Conforme sugerido na Figura 8, constata-se que a arquitectura deste serviço é semelhante à arquitectura adoptada no serviço de *email*.

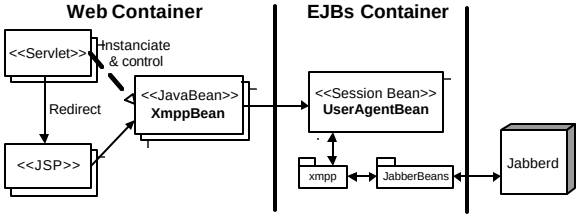


*Figura 8: Arquitectura do serviço de mensagens instantâneas e de presença.*

Durante a implementação deste serviço surgiu a dificuldade de criar a ilusão da recepção instantânea de mensagens e notificações de presença. Esta dificuldade surgiu na medida em que por um lado o servidor *Jabber* pode iniciar sessões de comunicação; mas, por outro lado, os clientes são browsers Web e como tal utilizam o protocolo HTTP que se baseia no padrão de comunicação pedido-resposta. Foram analisadas três soluções para ultrapassar tal dificuldade. Para qualquer uma delas são apresentadas também as suas desvantagens correspondentes.

**Polling**

Segundo esta estratégia o browser faz sistematicamente um refrescamento automático da página onde ocorrem as actualizações. Desta forma quaisquer alterações que ocorram no servidor manifestam-se na página refrescada. Esta solução tem a desvantagem de exigir um compromisso entre a sobrecarga na rede versus o tempo de actualização. Se se quer dar a noção de instantâneo diminui-se o tempo de actualização, contribuindo para o aumento de carga na rede; é uma desvantagem visto a maior parte dos refrescamentos não produzirem de facto quaisquer alterações.

**Resposta incompleta**

Nesta solução o servidor mantém o canal de resposta aberto. Desta forma assim que tiver actualizações a fazer, realiza-as através desse canal (pode enviar logo as actualizações ou provocar no cliente o refrescamento da página a actualizar). No entanto, para que este canal permaneça válido é necessário manter a *thread* que o criou, o que constitui uma desvantagem pelo facto de exigir um elevado número de recursos do lado do servidor Web; com a agravante de serem recursos que não são controlados pelo programador, mas sim pelo servidor, estando portanto dependente da implementação do mesmo.

**Applet**

Nesta solução, na página HTML produzida vai referenciada um applet que, não tendo apresentação visual, abre um canal de comunicação com o servidor, e ficando bloqueado à espera de notificações. Tem desvantagens ao nível da segurança e da administração de rede uma vez que o canal utilizado não é o porto 80. Por sua vez também gasta recursos do lado do servidor Web só que, ao contrário da solução anterior, gasta recursos que o programador controla.

As três técnicas foram implementadas embora apenas a última, applet, e a primeira, *polling*, tenham sido utilizadas na aplicação. A primeira solução é utilizada automaticamente se o browser não puder executar o applet Java.

Qualquer umas das três técnicas apresentadas são conhecidas conceptualmente como *client pull* [20] e *server push* [20]. A primeira técnica, *polling*, enquadra-se no *client pull*, ou seja, o servidor provoca que seja o cliente a originar a comunicação (através de *script* ou através do *header* do documento). As duas últimas técnicas, resposta incompleta e applet, enquadram-se no *server push*, onde é o servidor que, mantendo a ligação aberta com o cliente, toma a iniciativa de enviar para este último a informação actual.

**4.5 Integração de Serviços**

A integração dos serviços pode ser encarada de diferentes perspectivas: (1) integração ao nível da lógica de negócio, ou seja, ao nível da informação e funcionalidades disponibilizadas entre componentes; e (2) integração ao nível da apresentação. Ao nível da lógica de negócio, a partilha de informação é realizada através de objectos de sessão Web. Por outro lado, integração ao nível da apresentação (conforme ilustrado na Figura 9), exige que os vários serviços apareçam de forma coerente e transparente aos utilizadores finais.


*Figura 9: Interface Web do serviço myComs.*

A Figura 9 ilustra a interface Web do *myComs*, podendo-se verificar que existem cinco zonas perfeitamente distintas:

- topo e fundo, que são estáticas ao longo de toda a interacção do utilizador com o sistema;
- à esquerda, uma árvore de navegação no sistema, também ela estática;
- na zona central, a interface dedicada à apresentação do serviço de email; e por último
- à direita a zona dedicada ao serviço de mensagens instantâneas e notificações de presença.

Nesta última zona (à direita) pode-se verificar ainda que no seu topo permite que o utilizador altere o seu estado de presença; imediatamente abaixo verifique a sua lista de amigos, com os seus estados de presença respectivos; e por último, uma janela com as mensagens instantâneas recebidas. (Para o utilizador iniciar comunicação com um dos seus amigos basta premir sobre o nome do amigo correspondente.)

Esta divisão de apresentações foi conseguida recorrendo a *framesets* do HTML. Com esta técnica as actualizações, a acontecerem, serão parciais apenas ao nível do email ou ao nível das mensagens instantâneas e de presença não envolvendo a actualização da página total.

## 5. Arquitectura de *Clustering*

O projecto PUC visa, para além da construção de um protótipo funcional, explorar a possibilidade de fornecer o serviço recorrendo a máquinas com características convencionais (COTS, *Commercial Off-The-Shelf*). Assim, abordou-se uma arquitectura de suporte ao serviço baseada na utilização de *clusters*. Estes *clusters* são compostos por com um conjunto de máquinas interligadas por uma rede de comunicação de grande eficiência, e no qual se cria as condições necessárias ao fornecimento de serviços empresariais com uma elevada qualidade de serviço. Os requisitos impostos ao *cluster* e a arquitectura proposta para os satisfazer são descritos nesta secção.

Visto o PUC poder ser considerado como uma extensão dos serviços providenciados pelas empresas de telecomunicações, muito naturalmente se espera que a infra-estrutura de suporte do PUC partilhe as características das homólogas infra-estruturas de telecomunicações.

### 5.1 Requisitos Não Funcionais

**Escalabilidade**

O PUC deverá suportar um número de utilizadores subscritos na ordem dos milhões de utilizadores, ou seja, deve ser capaz de comportar a existência de milhões de contas de utilizador.

O serviço prestado deverá ter a capacidade de lidar com dezenas de milhar de acessos simultâneos.

Deverá ser possível escalar o número de utilizadores que o PUC suporta, quer em número total como em simultâneo. Tal deverá ser conseguido através da actualização do hardware disponível ou adicionando novas máquinas à infra-estrutura de suporte computacional. Neste segundo caso, o processo de actualização deve ser efectuado tanto quanto possível sem que se prejudique as restantes características do serviço.

**Tempo de resposta**

Os tempos de resposta da aplicação deverão ser inferiores à centena de milésimos de segundo. Este tempo de resposta deverá ser entendido como o tempo despendido pelos vários componentes do PUC, devendo-se ter em conta tempos de resposta de possíveis intervenientes externos (por exemplo, servidores de correio electrónico externos ou bases de dados remotas).

**Disponibilidade**

O serviço fornecido pelo projecto PUC é considerado um serviço que exige características de alta disponibilidade. A aplicação deverá estar disponível 99.999% do tempo, disponibilidade esta semelhante à exigida às aplicações da área de telecomunicações. Deverá ser possível efectuar manutenção do equipamento, com vista à melhoria das suas características (*upgrade*) ou reparação sem que se afecte o restante funcionamento do serviço. É tolerável a diminuição da eficiência do serviço durante o período de manutenção, mas este deve ser disponibilizado nesse período.

**Tolerância a faltas**

A arquitectura de suporte deve, tanto quanto possível, resistir a falhas dos seus componentes, tanto a nível de hardware como de software, sem que essas faltas comprometam o serviço prestado.

A recuperação de uma falta deve ser o mais transparente possível tanto para o utilizador como para o programador do serviço. Em particular, a arquitectura deverá recuperar automaticamente faltas

simples (como a sobrecarga de uma máquina), deverá permitir ao programador recuperar faltas mais complexas (que exijam conhecimento ao nível aplicacional) e somente na impossibilidade de recuperação nestes dois níveis deve o utilizador ter intervenção na recuperação da falta.

### 5.2 Arquitectura Tecnológica de Suporte

#### *Cluster* de Servidores

Tendo em conta as restrições impostas, a proposta de arquitectura para suporte do PUC consiste num *cluster* de servidores, numa arquitectura com uma única camada (i.e., todos os serviços correm no mesmo servidor físico conforme sugerido na Figura 10). O número de nós do *cluster* poderá ser incrementado até que se disponha do poder computacional, memória e disco necessário para lidar com o número de utilizadores existentes, satisfazendo assim o requisito de escalabilidade. A utilização de uma arquitectura de camada única é motivada pelo requisito de tempo de resposta reduzido, para o qual este tipo de arquitectura é o mais indicado, pois beneficia das optimizações de comunicação entre processos da mesma máquina virtual Java e evita a latência introduzida pela comunicação de rede.

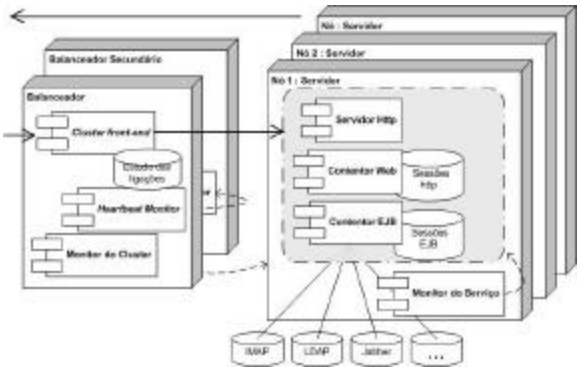


*Figura 10: Arquitectura de clustering proposta.*

Cada nó deste *cluster* deve assim providenciar um servidor HTTP, um contentor de servlets/JSP e um contentor de EJB, sendo estas as tecnologias que se utilizam no desenvolvimento do *myComs*. O contentor de EJB é providenciado por um servidor aplicacional J2EE [2], os quais em grande parte integram contentores de servlets/JSP, e até mesmo um servidor HTTP. Nesse caso, o servidor aplicacional será o único elemento necessário em cada nó.

#### Balanceamento de carga

O aumento do número de máquinas no *cluster* melhora o serviço ao possibilitar o balanceamento de carga entre os vários nós. Os pedidos devem ser distribuídos de uma maneira equilibrada e eficiente, aproveitando possíveis vantagens de localização. O balanceamento de carga deve ser abordado entre os vários níveis lógicos existentes: entre o cliente e o servidor HTTP, o servidor HTTP e o contentor de servlets e finalmente entre o contentor de servlets e o contentor de EJB. No entanto, como utilizamos uma arquitectura de camada única, a utilização de estratégias de colocação dos componentes invalida o balanceamento. Deste modo, apenas temos de nos preocupar com a distribuição de carga entre o cliente e os nós do *cluster*, nomeadamente os servidores HTTP existentes.

De modo a distribuir a carga pelos vários nós do *cluster*, opta-se por efectuar um balanceamento a nível do servidor, evitando os problemas inerentes à solução de balanceamento por *round-robin* DNS. Este balanceador é um elemento externo ao *cluster*, em hardware ou software, e deverá ter noção da falha de um nó de modo a reencaminhar os pedidos subsequentes para uma máquina disponível.

Devido ao grande número de ligações que é necessário suportar, o algoritmo é o mais simples possível, trabalhando a nível do IP (*layer-4*). Caso se verifique uma má distribuição de carga pelas máquinas o algoritmo deverá entrar em conta com o número de ligações abertas em cada nó. A utilização de balanceamento em função do conteúdo (ou seja, a *layer-7*) é evitada por introduzir carga e latência que se deseja evitar.

Embora o balanceamento seja feito a *layer-4*, não se põe de parte que haja vantagens em direccionar pedidos do mesmo utilizador para o mesmo nó. Assim, o balanceador inclui persistência de sessão baseada no mapeamento temporário entre o cliente e o servidor seleccionado.

#### Replicação de estado

O balanceamento proposto só pode ser efectuado de modo tão simples como descrito anteriormente se todos os nós do *cluster* tiverem as mesmas capacidades (funcionais) para responderem a qualquer pedido. Visto que o processamento dos pedidos feitos ao *myComs* inflige mudanças no estado aplicacional, essas alterações têm de ser repercutidas nos restantes nós. Na arquitectura proposta as várias instâncias de servidores HTTP e servidores aplicacionais J2EE replicam o estado das sessões abertas entre si, permitindo por um lado a utilização do balanceamento referido e por outro evitando que os servidores se tornem pontos únicos de falha.

Esta replicação de estado deve ser feita em memória, ao invés de num sistema de ficheiros partilhado ou base de dados centralizada. A motivação para esta decisão é a diminuição do tempo de acesso ao estado, a diminuição do impacto da perca de um servidor que mantenha o estado partilhado e a possibilidade de utilizar os recursos disponíveis em

todo o *cluster* ao invés de uma super máquina que suporte todo o estado do *cluster*.

A replicação deve incidir sobre o estado das sessões HTTP e o estado dos *statefull session* EJB, pois estes são os mecanismos de manutenção de estado do *myComs* ao longo das diversas interacções do utilizador. O protocolo de comunicação das mensagens de replicação assenta num mecanismo de *multicast*, permitindo a sincronização do estado entre os vários nós com uma única mensagem, evitando tráfego desnecessário.

**Replicação do balanceador**

O balanceador de carga é apresentado como um elemento único na arquitectura, pelo que é à partida um ponto único de falha. Como representado na Figura 10, este deve estar replicado, garantindo a continuação do serviço em caso da sua falha. Para rentabilizar a existência de ambos os balanceadores, poder-se-á utilizar um nível extra de indirecção, por exemplo através da utilização de *round-robin* DNS, mas nesta situação deve ser possível a cada um dos balanceadores tomarem o lugar do outro em caso de falhas. Para isso utiliza-se uma técnica designada de *ARP spoofing*, que permite a obtenção de um IP existente noutra máquina (vulgarmente conhecido como "*IP-Takeover*"), sendo necessário que as interfaces de rede dos balanceadores suportem a existência em simultâneo de dois endereços.

O estado das sessões mantido nos balanceadores de carga são replicadas para a unidade redundante ou armazenadas num sistema de ficheiros partilhado de modo a suportar a recuperação sem perca de sessões a nível do utilizador.

### 5.3 Implementação

Cada nó do cluster é composto por um servidor J2EE Jboss, o qual integra um servidor HTTP e contentor Web com um contentor de EJB na mesma máquina virtual Java. O Jboss conta desde a versão 3.0.0 com suporte para *clustering*, em particular o suporte de replicação em memória do estado de sessões HTTP e EJB, a qual é necessária para a recuperação de faltas no *cluster*. Este mecanismo mantém o estado de sessão uniforme em todos os nós do *cluster* tornando-os aptos a responder a qualquer pedido. A replicação utiliza uma extensão da comunicação *multicast* para grupos designada por JavaGroups [15] e é desencadeada pela alteração de uma variável contida numa sessão.

Para efectuar o balanceamento de carga pelos nós do cluster, adoptou-se a solução providenciada pelo projecto LVS (*Linux Virtual Server*) [16]. Este projecto consiste num *patch* para o kernel do sistema operativo Linux, o qual permite criar serviços "virtuais": os pacotes recebidos no balanceador são encaminhados para os nós do cluster de servidores, os quais efectivamente processam o pedido e devolvem a resposta. O cliente apenas vê o balanceador e tem a noção que é este que providencia o serviço. O encaminhamento é feito a nível de TCP/IP, ou seja, o LVS pode ser considerado um *switch* de *layer-4* por software.

O balanceador foi configurado no modo LVS-DR, funcionando assim com encaminhamento *one-way packet-fowarding*[19]. Este apresenta menor latência, suporta um maior número de máquinas no *cluster* e possibilita a utilização de nós em vários sistemas operativos. O algoritmo de balanceamento é *round-robin* com persistência temporária: o utilizador é direccionado para uma mesma máquina durante um intervalo de tempo, de modo a aproveitar possíveis vantagens de *caching* do estado de sessão no servidor.

A tolerância e recuperação de faltas é providenciada a dois níveis: nos nós do *cluster* e no balanceador. Em primeiro lugar, qualquer nó do *cluster* pode responder a um pedido da interacção entre um cliente e um nó que falhou, recorrendo ao estado de sessão replicado. No entanto, para que esta recuperação tenha lugar é necessário que os pedidos deixem de ser enviados para o nó em falta e passem a ser encaminhados para um nó "saudável". Esta mudança no encaminhamento é conseguida a nível do LVS, por monitorização dos vários nós do *cluster*. Aquando da detecção da falha de um nó, o monitor informa o LVS que o nó deixou de estar disponível. Nesta altura, o LVS retira esse nó da sua lista de nós activos e passa a redireccionar os pacotes para um nó alternativo. Na nossa solução todos os nós mantêm uma replicação total do estado do *cluster* pelo que é indiferente qual o nó que substitui o que falhou.

## 6. Conclusões

Este artigo introduz a motivação, enquadramento, princípios e visão geral do projecto "PUC – Sistema de Comunicações Pessoais para as Redes de Próxima Geração". Atente-se que os objectivos gerais do projecto PUC são bastante ambiciosos e o seu desenho e implementação não trivial.

Alguns dos princípios enumerados na Secção 2.1 foram relativamente analisados neste artigo, em particular (1) o princípio da adopção de tecnologia Java e de projectos *open-source*; e o princípio do suporte ao elevado número de utilizadores e de transações.

Relativamente ao primeiro princípio, é de salientar que embora a tecnologia usada (baseada no J2EE) seja actualmente madura e estável, existe uma "curva de aprendizagem" extremamente abrangente e acentuada, o que passou necessariamente pela

aprendizagem e domínio de várias especificações agregadas ao J2EE (e.g., Servlets, JSP, Java Mail, JNDI, EJB, JDBC, JAXR) para além da integração, modificação e ou configuração dos produtos *open source* usados (e.g., Jboss, JWMA, JabberBeans).

Relativamente ao segundo princípio, é de referir que a arquitectura de *clustering* proposta e discutida na Secção 5 encontra-se actualmente em fase de testes de carga e de desempenho relativamente aos requisitos identificados de escalabilidade, disponibilidade e de tolerância a faltas.

Embora, por motivos de simplicidade, a apresentação neste artigo se tenha centrado no serviço *myComs*, encontra-se em fase de desenvolvimento os restantes serviços referidos, *myContacts* e *myAgenda*, de forma integrada com o anterior. Adicionalmente, encontra-se também em desenvolvimento versões de aplicações (correspondentes aos serviços referidos) para serem pré-instaladas e executadas em terminais móveis sobre o ambiente J2ME (*Java 2 Micro Edition*).

O trabalho futuro passará (1) pela integração ao nível dos componentes de software e ou ao nível da camada de apresentação dos vários serviços desenvolvidos; (2) pela integração de funcionalidades associadas ao conceito de localização geográfica e de presença; (3) pelo desenho e realização de testes e optimizações à arquitectura de modo a garantir os requisitos definidos; e (4) pelo suporte de novos formas de interacção homem-máquina (e.g., baseada no VoiceXML).

## Referências


[1] INESC-ID, Grupo de Sistemas de Informação. *The PUC Project*. http://berlin.inesc-id.pt/projects/puc/

[2] Especificação J2EE, http://java.sun.com/j2ee

[3] Especificação J2ME, http://java.sun.com/j2me

[4] Servidor de Aplicações J2EE Jboss, http://www.jboss.org

[5] Open Source Initiative, http://www.opensource.org/

[6] SourceForge Site, http://sourceforge.net

[7] JWMA (*Java Web Mail Architecture*), http://jwma.sourceforge.net/

[8] Protocolo Jabber, http://www.jabber.org

[9] Biblioteca JabberBeans, http://www.jabberstudio.org

[10] *Enterprise Java Beans* (*EJBs*), http://java.sun.com/products/ejb/index.html

[11] *JavaMail API*, http://java.sun.com/products/javamail/index.html

[12] *Microsoft Messenger*, http://www.msn.com

[13] *I Seek You* (*ICQ*), http://web.icq.com/

[14] *American Online IM* (*AIM*), http://www.aim.com/

[15] Protocolo de comunicação multicast para grupos – JavaGroups, http://www.javagroups.org

[16] Projecto *Linux Virtual Server*, http://www.linuxvirtualserver.org

[17] G. Grasner, S. Pope. "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80". *Journal of Object-Oriented Programming*, 1 (3), 1988.

[18] M. Juric, et al. *J2EE Design Patterns Applied*. Wrox Press, 2002.

[19] V.Cardellini, E. Casalicchio, M. Colajanni, P.S. Yu, ``The state of the art in locally distributed Web-server systems", *ACM Computing Surveys*, Vol. 34, No. 2, June 2002.

[20] An Exploration of Dynamic Documents, http://wp.netscape.com/assist/net_sites/pushpull.html

[21] "Mensageiro" do Portal SAPO, http://mensageiro2.sapo.pt

[22] XMPP (*eXtensible Messaging and Presence Protocol*), http://www.jabber.org/ietf/

[23] VoiceXML, http://www.voicexml.org/