Comunicações Pessoais Unificadas

João Pedro Guerreiro da Graça Patriarca
Alberto Rodrigues da Silva
CCISEL, Instituto Superior de Engenharia de Lisboa,
Rua Conselheiro Emídio Navarro, 1, 1949-014 Lisboa,
jpatri@cc.isel.ipl.pt
INESC-ID, Instituto Superior Técnico,
Rua Alves Redol, 9, 1000-029 Lisboa,
alberto.silva@acm.org

O projecto "PUC – Sistema de Comunicações Pessoais para redes de próxima geração" integra, numa mesma plataforma, um conjunto de serviços de comunicação pessoal, nomeadamente, os serviços de correio electrónico e o de mensagens instantâneas. A plataforma aplicacional de suporte é o JBoss que segue a especificação J2EE. Os serviços foram implementados de forma a garantirem a independência do tipo de terminal de acesso (podendo o acesso ser via terminal Web, voz, WAP ou outro). Este artigo apresenta o trabalho desenvolvido segundo, principalmente, uma perspectiva de engenharia, focando os detalhes de implementação que tiveram maior relevância, nomeadamente a descrição dos problemas, das soluções encontradas e implementadas, discutindo vantagens e desvantagens das soluções adoptadas, e finalmente sugerindo soluções alternativas.

Palavras-Chave: Comunicações Pessoais Unificadas; Gestão de Informação Pessoal; Integração de serviços; Código livre; Java; J2EE

1 Introdução

As operadoras de telecomunicações têm vindo progressivamente, e em todo o mundo, a substituir as suas redes antigas (analógicas e ou digitais, orientadas fundamentalmente para serviços de voz baseados na telefonia) por redes de nova geração, onde se estima que os serviços de dados e multimédia ultrapassem, em termos de tráfego da rede, os tradicionais serviços de voz.

Para sustentar os enormes investimentos financeiros realizados pelas operadoras de telecomunicações é necessário criar urgentemente novos serviços por forma a manter e angariar novos clientes. É nesse sentido que surgiu este trabalho. Mostrar a capacidade de concepção e realização de serviços inovadores que respondam a necessidades actuais e futuras.

O "PUC - Sistema de Comunicações Pessoais para as Redes de Próxima Geração" foi 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.

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"*). O principal factor de inovação do projecto PUC reside no facto destes serviços serem 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 trabalho realizado pelo autor deste artigo foi desenvolver e implementar o protótipo correspondente aos serviços na área das comunicações pessoais com interface *Web*, denominado por *"myComs"*. Teve também a responsabilidade de avaliar e explorar arquitecturas tecnológicas emergentes e arquitecturas de software de suporte, nomeadamente tecnologias na área J2EE (*Java 2 Enterprise Edition*).

Este artigo foca principalmente a vertente de engenharia e está organizado em 6 secções. A secção 1 introduz o contexto e motivação deste projecto. A secção 2 apresenta uma visão geral da arquitectura tecnológica do sistema e um conjunto de requisitos a que o sistema deve

obedecer. A secção 3 esclarece a arquitectura de software adoptada no desenho dos serviços. A secção 4 discute problemas, detalhes de desenho e implementação do sistema *"myComs"*. A secção 5 critica algumas soluções adoptadas e apresenta soluções alternativas. Por último a secção 6 corresponde às conclusões finais.

2 Visão Geral

A figura 1 ilustra a visão geral da arquitectura tecnológica 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 Correio Electrónico, 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.

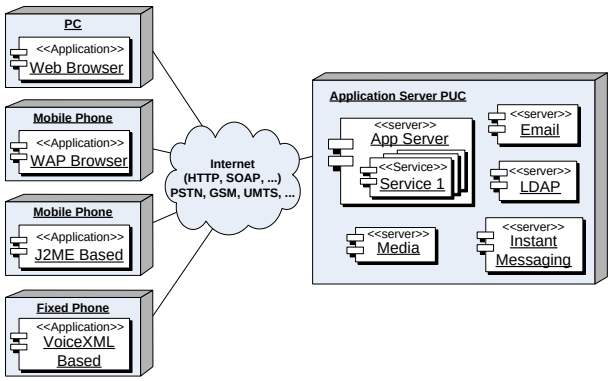


Figura 1: Visão geral da arquitectura tecnológica do sistema PUC

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). 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. Por outro lado 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).

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. O sistema deve ainda 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).

Como referido anteriormente o trabalho desenvolvido pelo autor focou-se nos serviços de comunicação pessoais com interface *Web*. Dentro das comunicações pessoais foram eleitos alguns serviços considerados de maior utilização na actualidade: (1) serviço de correio electrónico; (2) serviço de mensagens instantâneas e notificação de presença; (3) considerado também um serviço o serviço de subscrição e autenticação.

2.1 Serviço correio electrónico

Um serviço de *webmail* é um serviço que permite ao utilizador aceder ao seu correio electrónico através de um *browser*. O serviço de correio electrónico deverá permitir aos seus utilizadores várias funcionalidades, de entre as quais se destacam: a possibilidade de envio e recepção de mensagens de correio electrónico com ou sem *attachement*; a gestão de caixas de correio (e.g., segundo o paradigma de pastas hierárquicas) - com esta funcionalidade o utilizador consegue criar novas pastas, remover pastas, apagar mensagens, mover mensagens entre pastas; 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 correio electrónico, foi adoptado e adaptado como projecto de base o sistema código livre JWMA [10].

2.2 Serviço de mensagens instantâneas e de presença

O serviço de mensagens instantâneas e presença é um serviço que cumpre os mesmos objectivos que os serviços proprietários da Microsoft (Messenger), I Seek You (ICQ), da AOL (AIM), entre outros. Entre as várias funcionalidades permite: alterar o estado de presença do utilizador; manter uma lista de amigos com a possibilidade de estar organizada por grupos e com o estado de presença de cada um; enviar mensagens instantâneas para um amigo cujo estado de presença seja presente. Uma das mais valias deste serviço é ser disponibilizado com interface *Web*, requerendo ao utilizador, no máximo, um *browser*. É uma mais valia devido à dificuldade da sua implementação porque, ao contrário de uma aplicação *Web* normal em que por norma a aplicação se baseia em pedido/resposta, aqui, neste serviço, as notificações são assíncronas.

Este serviço em relação ao serviço de correio electrónico tem um número de acções limitado disponibilizando apenas o essencial: receber mensagens e notificações de presença; enviar mensagens e alterar estado de presença. Tal facto deve-se a ter sido um serviço desenvolvido de raiz; ao contrário do serviço de correio electrónico que tinha já previsto todas as funcionalidades referidas. A troca de mensagens neste serviço obedece o protocolo *Jabber* [6].

2.3 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. Neste sentido a subscrição foi entendida como mais um serviço do *"myComs"*, reflectindo-se logo na arquitectura de software adoptada que segue o mesmo modelo dos serviços correio electrónico e mensagens instantâneas. 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).

3 Arquitectura de Software

A arquitectura e implementação dos vários serviços seguiu a especificação J2EE (Java 2 Enterprise Edition) [4]. O J2EE é um standard, que utiliza a linguagem de programação Java, vocacionado para o desenvolvimento e distribuição de aplicações empresariais orientadas para a *Web*. Por ser standard não está comprometido com um fabricante, e para além disso, os contentores onde a aplicação irá executar já fornecem um conjunto de serviços de sistema que de outra forma seriam da responsabilidade do programador, como por exemplo, serviços a nível de segurança, transacções, gestão do ciclo de vida dos componentes, gestão de *threads*, entre outros.

Conforme ilustrado na figura 2, a arquitectura adoptada para o desenho dos serviços baseia-se numa arquitectura três camadas: (1) camada de acesso à informação; (2) camada correspondente

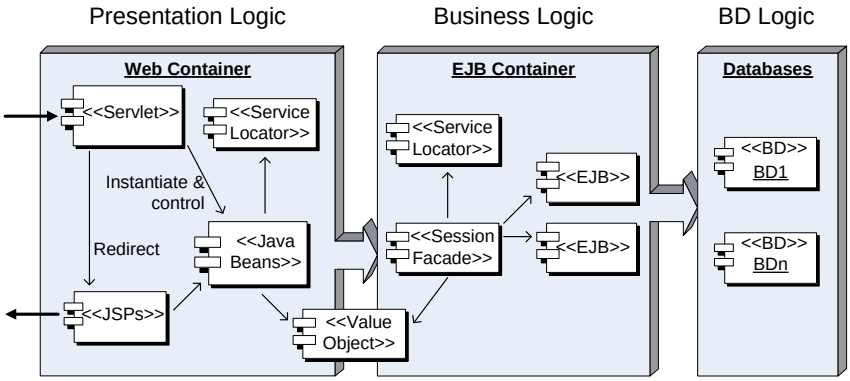


Figura 2: Arquitectura geral de um serviço

à lógica de negócio; (3) camada correspondente à apresentação. Este modelo é largamente utilizado em aplicações do lado do servidor com interface *Web*.

No desenho da arquitectura dos serviços foram adoptados alguns padrões de desenho J2EE [1] entre os quais se destacam os padrões *Service Locator*, *ValueObejct* e *Session Facade*. Na componente *Web* a arquitectura seguiu igualmente o padrão de desenho MVC (*Model-View-Controller*) onde as *Servlets* desempenham o papel de controlo, os *JavaBeans* o papel de modelo, e as *Java Server Pages* desempenham o papel de apresentação.

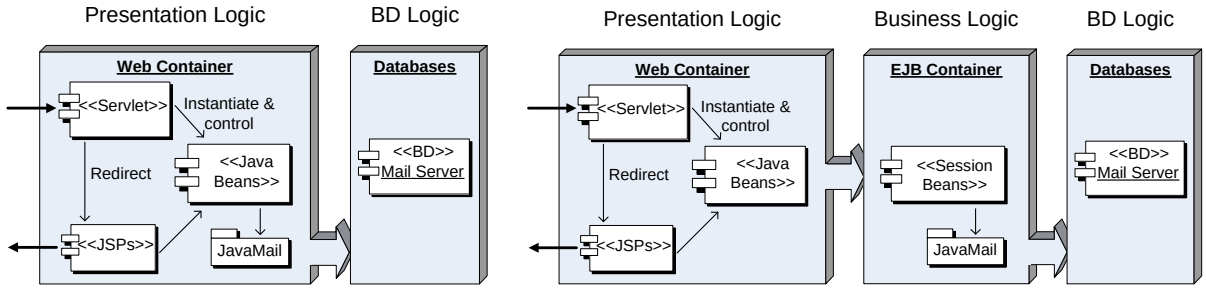
4 Problemas e Soluções de Desenho

Nesta secção, dado o limite de páginas para o artigo, foram seleccionados alguns pormenores e decisões de implementação que se acharam de maior relevância.

4.1 Correio electrónico

De forma a integrar no *"myComs"* o serviço de correio electrónico, foi adoptado e adaptado como projecto de base o sistema JWMA [10], que tem a característica de ser código livre.

O JWMA original baseia-se numa arquitectura *Web* simples, de duas camadas (servidor *Web* + servidor de correio electrónico). A figura 3(a) ilustra a arquitectura correspondente ao JWMA. Esta opção 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 (figura 3(b)).



(a) Versão original

(b) Versão adaptada

Figura 3: Arquitectura do serviço de correio electrónico

A adaptação correspondeu a introduzir uma segunda camada entre o contentor *Web* e o servidor de correio electrónico, designado por contentor de EJBs (*Enterprise Java Beans*) e migrar para os novos componentes criados toda a lógica de negócio existente na versão original nos *JavaBeans*. O acesso ao servidor de correio electrónico passou a ser da responsabilidade dos componentes aí residentes, designados por EJBs. A utilização de EJBs respondeu 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 EJBs disponibilizam interfaces remotas permitindo uma fácil utilização por outros componentes; é da responsabilidade do contentor de EJBs um conjunto de serviços de sistema, nomeadamente, suporte a transacções e segurança libertando o programador dessa mesma responsabilidade; para permitir ainda, entre outras características, escalabilidade, o contentor de EJBs dá suporte à criação de um ambiente distribuído com a criação de *clusters*.

Para uma melhor compreensão das alterações realizadas, a figura 4 exemplifica a adaptação feita ao nível dos *JavaBeans*. Está ilustrada apenas uma parte da arquitectura uma vez que, para os restantes elementos, o processo foi idêntico, ou seja, para cada *JavaBean* existe um EJB correspondente.

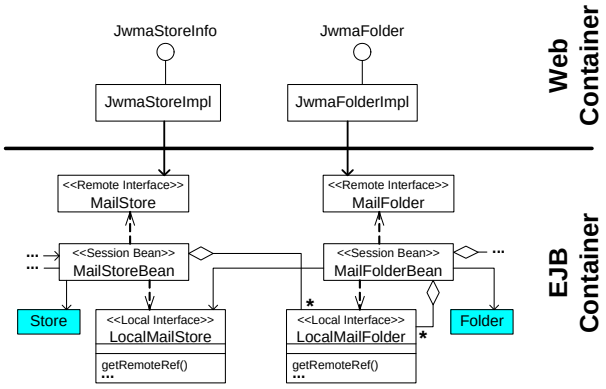


Figura 4: Exemplo de adaptação dos JavaBeans a EJBs

Em relação aos *JavaBeans* foram mantidas as interfaces, o que permitiu não ter que se alterar as classes que os usavam (*Servlets* e JSPs), e foi-lhes acrescentado referências remotas para os respectivos EJBs. 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 EJBs. Foram utilizadas interfaces locais por duas razões: primeiro, e admitindo que os vários EJBs não têm sentido executar-se em contextos diferentes, ganha-se em eficiência usando interfaces deste tipo; segundo, porque na interacção entre EJBs são passados como parâmetros instâncias pertencentes à API do `JavaMail`. 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. Esta interface é conseguida através do atributo `sessionContext`, atributo este que é afectado ao componente durante o processo de criação do mesmo. Este factor não seria importante se tratasse-se de componentes empresariais sem estado (*stateless session EJB*); acontece que estes componentes são componentes *statefull* que têm como característica ficarem afectos a um mesmo cliente durante toda a sessão, e como tal ambas as referências, locais e remotas, devem referenciar a mesma instância.

4.2 Mensagens instantâneas

Para implementação deste serviço foi utilizada a biblioteca Java código livre *JabberBeans*, catalogada, tal como o JWMA, no portal *sourceforge*[1]. Esta biblioteca implementa o protocolo *Jabber* [6]. É uma implementação de baixo nível e apenas abstrai a criação de tramas XML, que são o suporte do protocolo para comunicação entre cliente e servidor. Devido a este facto foi desenvolvida a 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.

A arquitectura deste serviço é semelhante à arquitectura adoptada no serviço de correio electrónico, seguindo o modelo de três camadas.

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, por outro, 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 esta dificuldade: (1) *Polling*; (2) Resposta incompleta; (3) Applet. Para qualquer uma delas são apresentadas igualmente as desvantagens correspondentes.

4.2.1 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, o que é uma desvantagem visto a maior parte dos refrescamentos não produzirem, de facto, quaisquer alterações. No campo visual, esta solução tem a desvantagem de causar incómodo ao utilizador por este se aperceber do carregamento cíclico da página.

4.2.2 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. Igualmente no campo visual, esta solução tem a desvantagem de não terminar a barra de carregamento da página.

4.2.3 Applet

Nesta solução, na página HTML produzida é referenciada uma *Applet* que, não tendo apresentação visual, abre um canal de comunicação com o servidor, ficando bloqueada 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 ensaiadas 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 a *Applet* Java. Qualquer umas das três técnicas apresentadas são

[1] Portal com o objectivo de divulgar projectos de cariz código livre - http://sourceforge.net/

conhecidas conceptualmente como *client pull* [8] e *server push* [8]. A primeira técnica, *polling*, enquadra-se no *client pull*; as duas últimas técnicas, resposta incompleta e *Applet*, enquadram-se no *server push*.

5 Discussão do Trabalho

Neste artigo, na secção 4 foram apresentados dois pormenores de implementação considerados dos que tiveram maior relevância na realização deste trabalho. Nesse sentido, nesta secção são apresentadas soluções alternativas apenas para as soluções descritas.

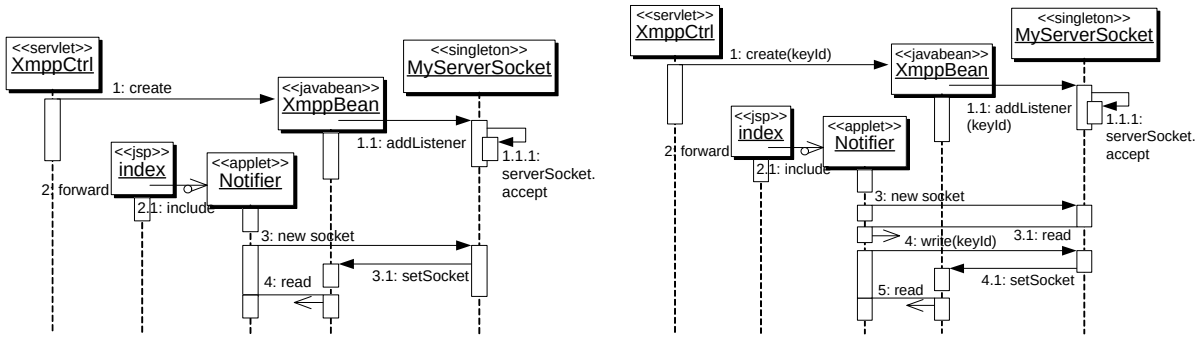
5.1 Correio electrónico

Na solução implementada todos os *JavaBeans* têm um componente empresarial EJB correspondente. Esta solução é uma consequência do facto de muita lógica do serviço de correio electrónico existir ainda na componente *Web*, nomeadamente lógica correspondente à hierarquia de entidades definidas na própria API do JavaMail. Com esta solução, um cliente deste serviço é obrigado a conhecer bem todos os componentes que constituem a interface pública deste mesmo serviço. Uma proposta alternativa seria existir um único componente remoto que representasse o serviço correio electrónico, cabendo a ele a responsabilidade de conhecer e utilizar apropriadamente os restantes componentes existentes. Esta solução corresponde a utilizar o padrão de desenho perfeitamente catalogado denominado por *Session Facade*. Neste cenário o cliente precisa apenas conhecer um componente e a sua interface remota. Como aspecto negativo desta alternativa, a interface remota deste componente tornar-se-ia mais pesada.

5.2 Mensagens instantâneas

Para este serviço, dadas as suas características assíncronas e dada a interface do utilizador[2], fica complicado arranjar uma solução conceptualmente válida. Foram apresentadas três possíveis soluções e qualquer uma das três soluções apresenta aspectos positivos e aspectos negativos. Nesse sentido, não se propõe aqui uma nova solução para o serviço; em vez disso, sugere-se uma correcção à implementação na solução adoptada.

Esta solução exigiu na componente *Web* a criação de várias entidades com distintas responsabilidades, conforme ilustrado na figura 5(a): (1) uma entidade que implementa o padrão *Singleton* (`MyServerSocket`) responsável por centralizar e atender pedidos de novas ligações; (2) um *JavaBean* (`XMPPBean`) que representa uma sessão e que é responsável por manter a nova ligação criada para este cliente; (3) uma *Applet* (`Notifier`) que fica igualmente associada à nova ligação e que é responsável por processar notificações da recepção de mensagens assíncronas.



(a) Solução implementada

(b) Solução alternativa

Figura 5: Sequência de acções na criação do canal de comunicação

[2]Neste protótipo foi apenas desenvolvida a interface homem–máquina PC com interface *Web*.

Durante o processo de estabelecimento do canal de comunicação foi necessário criar um identificador que associasse o pedido de um canal de comunicação à *Applet* e ao *JavaBean*. O identificador utilizado correspondeu ao endereço IP do cliente. Não é razoável utilizar esta informação como identificador porque facilmente se consegue arranjar cenários onde diferentes pedidos de ligação podem ser vistos com o mesmo endereço IP, como por exemplo, pedidos originados numa VPN (*Virtual Private Network*).

Uma possível solução para permitir cenários deste tipo passa por responsabilizar a *servlet* (`XMPPCtrl`) de criar um identificador único dando-o a conhecer à *Applet* e ao *JavaBean*. O *JavaBean* utilizava desta feita este identificador único para se registar no *Singleton*. Era necessário criar um protocolo simples para troca de identificadores entre a *Applet* e o *Singleton* por forma ao *Singleton* conseguir associar o novo canal criado ao *JavaBean* respectivo. A figura 5(b) é ilustrativa da sequência de acções necessárias para a resolução deste problema.

6 Conclusões

O suporte tecnológico utilizado neste projecto – desde especificações (e.g. *J2EE*, *LDAP*, *Jabber*), como ferramentas para desenvolvimento, instalação e realização deste projecto (e.g. *netbeans*, *JBoss*, *openLdap*, *jabberd*, *qmail*, *courier*, ...) – seguiu um requisito comum, que foi o facto de ser utilizado ferramentas e tecnologia baseadas na abordagem de "código livre". Esta característica, apesar de oferecer as vantagens reconhecidas, implicou contudo alguns problemas, designadamente: (1) documentação fraca e muito dispersa; (2) ferramentas de desenvolvimento de software e de administração dos servidores fracamente integradas e pouco produtivas. Estes factores implicaram uma curva de aprendizagem lenta e pouco produtiva. No entanto, verificou-se que após o conhecimento adequado da tecnologia, o desenvolvimento de aplicações distribuídas ficou facilitado dado o conjunto de serviços base que a plataforma disponibiliza.

O projecto PUC foi concretizado por vários grupos de trabalho com objectivos concretos, nomeadamente: (1) o descrito neste artigo e dedicado às comunicações e que teve como consequência uma tese de mestrado [9]; (2) análise e avaliação de requisitos não funcionais do sistema [2]; (3) implementação dos serviços de gestão de informação (contactos e agenda) com interface *Web* [5]; (4) disponibilização de interfaces de utilizador móvel e voz [3]; (5) integração de todos os serviços desenvolvidos [7]. Estes trabalhos deram origem aos respectivos TFCs (Trabalhos Finais de Curso).

Referências

[1] Deepak Alur, Jonh Crupi, and Dan Malks, *Core j2ee patterns*, Sun Microsystems Press, 2001.

[2] João Clemente, *Personal unified communication - análise e avaliação de requisitos não funcionais*, Tech. report, Instituto Superior Técnico, Inesc-ID, 2003.

[3] Nuno Domingos and Nuno César, *Projecto puc - "personal unified communication" - sistemas embebidos para o contexto puc*, Tech. Report 149, Instituto Superior Técnico, Inesc-ID, 2003.

[4] Java Community Process (JCP) expert group (JSR-58), *JAVA^{TM} 2 platform, enterprise edition 1.3 specification*, Tech. report, Sun Microsystems, Setembro 2001, http://www.jcp.org/en/jsr/detail?id=58.

[5] Marco Guimarães and Rui Maia, *Personal unified communications*, Tech. report, Instituto Superior Técnico, Inesc-ID, 2003.

[6] *Protocolo xml standard para troca de mensagens na internet*, http://www.jabber.org/jeps/.

[7] David Martins and Eurico Frade, *Personal unified communication - integração*, http://berlin.inesc.pt/alb/uploads/1/189/PUC3-Relatorio-Final.pdf, 2004.

[8] Netscape Navigator, *An exploration of dynamic documents*, http://wp.netscape.com/assist/net_sites/pushpull.html/.

[9] João Pedro Patriarca, *Puc - sistema de comunicações pessoais para redes de próxima geração*, Tech. report, Instituto Superior Técnico, Inesc-ID, Maio 2005.

[10] Dieter Wimberger and Leonard Sitongia, *Implementação de um cliente web para acesso a uma conta de mail*, http://jwma.sourceforge.net/.