CENTRO DE EDUCAÇÃO PROFISSIONAL DE TIMBÓ – CEDUP TIMBÓ
CURSO TÉCNICO EM INFORMÁTICA COM HABILITAÇÃO EM
DESENVOLVIMENTO DE SOFTWARE
III MÓDULO – 2014/1

PROJETO INTERDISCIPLINAR: PROTÓTIPO DE UM SOFTWARE PARA COMUNICAÇÂO INSTANTÂNEA EMPRESARIAL EM JAVA

Roberto Luiz Debarba

RESUMO

O protótipo apresentado nesse artigo tem por objetivo proporcionar uma solução para maior velocidade de comunicação em empresas e instituições, apresentado um software de comunicação instantânea empresarial focado em velocidade, segurança e restringido seu alcance ao ambiente profissional. Após observação de diversos ambientes corporativos, identificou-se na maior parte comunicação e dinamismo falho entre seus colaboradores, onde era necessário locomover-se pela empresa ou utilizar um telefone para comunicação, mesmo que está se restringia à troca de poucas palavras. O sistema OpenLync oferece uma solução gratuita e de código aberto à resolução dos problemas apresentados. O baixo custo de implantação se reflete vantajoso à empresas de pequeno porte, onde a compra de outros sistemas pagos do gênero, como Microsoft Lync, não é financeiramente viável. Em muitos ambientes são utilizadas ferramentas gratuitas, como Google Chat ou Microsoft Skype, porém estas permitem a comunicação com pessoas não ligadas ao ambiente de trabalho, como amigos e familiares, diminuindo significativamente a produtividade geral. A ferramenta fornece uma aplicação de utilização fácil e rápida, onde os usuários podem trocar mensagens com colegas de trabalho, enviar recados à pessoas desconectadas e adicionar contatos de interesse à sua lista de amigos, facilitando a utilização no dia a dia. É apresentado ao usuário o nome completo de seus contatos, junto ao cargo e foto de perfil, melhorando a identificação de pessoas de menor contato. A manutenção dos usuários é feita por um profissional de tecnologia, que de forma facilitada pode cadastrar, editar ou remover funcionários conforme a necessidade da empresa.

Palavras-chave: Comunicação; Dinamismo; Empresarial.

1 ESTUDO DE CASO

Uma empresa de sistemas necessita melhorar a velocidade de comunicação entre seus funcionários. O sistema de e-mails é pratico e confiável, porém em diversos casos é necessária uma resposta instantânea. A empresa possui cerca de oitocentos funcionários operando em sua matriz, mais quatro filiais espalhadas pelo país com cinquenta colaboradores cada.

Para contornar o problema, os profissionais de TI decidiram implantar nas estações de trabalho um comunicador instantâneo, que proporcionará maior agilidade nos processos que exigem integração entre os envolvidos. No mercado é possível encontrar dois tipos de opções. Inicialmente, a solução que oferece maior número de benefícios é focada em ambientes

corporativos, como o Microsoft Lync, porém este possui um valor elevado, o que torna sua implantação inviável. A segunda opção é a utilização de um comunicador gratuito, como os já conhecidos Microsoft Skype e Google Chat. Pelo baixo custo de implantação, a opção foi amplamente estudada, mas após uma pesquisa em diversas empresas que adotaram as ferramentas, chegou-se a conclusão que a possibilidade dos funcionários se comunicarem com pessoas sem ligação com a empresa, como parentes e amigos, diminuiria a produtividade geral.

Sem mais alternativas disponíveis, a empresa decidiu apoiar o desenvolvimento de um aplicativo aberto que utiliza dados mantidos sob seu controle. O sistema seria implantado sem custos de licença e apenas funcionários teriam acesso.

O sistema de comunicação deve permitir que os funcionários acessem a ferramenta através de um nome de usuário e senha. No banco de dados devem ser cadastrados o nome completo de cada funcionário, seu cargo na empresa juntamente com sua imagem de perfil, para facilitar a identificação. A base de dados com todas as informações dos usuários deve estar em posse da empresa, limitando a comunicação à níveis internos e proporcionando maior segurança da informação, sendo que terceiros, como os provedores de serviços Google e Microsoft ou o governo não tenham acesso às informações.

Os usuários somente podem acessar o aplicativo para comunicar-se, onde o cadastro, edição e exclusão de colaboradores fica a cargo do setor de tecnologia, que irá administrar o servidor.

É possível ao usuário adicionar os contatos de maior interesse à uma lista de amigos e acessar o histórico de conversas anteriores.

Os dados necessários ao sistema, como informações dos usuários e mensagens devem trafegar e ser mantidos de forma criptografada, evitando ataques por escaneamento de rede ou roubos de informação do servidor.

1.1 REQUISITOS FUNCIONAIS

- RF01 – O sistema deve manter cadastro de Usuários:
  - Código;
  - Nome completo;
  - Nome de usuário;
  - Senha;
  - IP;
  - Cargo;
  - Imagem de perfil;
- RF02 – O sistema deve manter cadastro de Cargos:
  - Código;
  - Nome do cargo;
- RF03 – O sistema deve manter o histórico das mensagens enviadas:
  - Código;
  - Conteúdo da mensagem;
  - Data e hora do envio;
  - Remetente;
  - Destinatário;
  - Estado da mensagem(lida ou pendente);
- RF04 – O sistema deve manter uma lista de contatos de cada usuário;
- RF05 – O sistema deve possuir dois níveis de acesso:

    - Administrador;
    - Usuário;
- RF06 – Os sistema deve possuir três módulos:
    - Aplicativo cliente;
    - Aplicativo de gerenciamento de cadastros;
    - Serviço de controle de mensagens;
- RF07 – O aplicativo cliente deve permitir a comunicação entre os usuários;
- RF08 – O aplicativo de gerenciamento de cadastros deve permitir a manutenção dos cadastros de cargos e usuários;
- RF09 – O serviço ("daemon" no Linux) de controle deve gerenciar o envio e recebimento de mensagens entre os módulos cliente;
- RF10 – No aplicativo de gerenciamento de cadastros deve ser possível *logar* apenas com uma conta de "administrador", enquanto no cliente é possível realizar o *login* com qualquer tipo de conta;
- RF11 – Ao abrir o sistema(exceto ao iniciar/parar o serviço) é necessário entrar com nome de usuário e senha para identificação do colaborador;
- RF12 – Os usuários podem acessar o histórico de suas conversas;
- RF13 – Os usuários podem manter múltiplas janelas de conversação simultâneas;
- RF14 – Os usuários possuem a opção de exibir apenas contatos *online*;
- RF15 – Ao enviar uma mensagem para um usuário *offline*, o sistema deve marcá-la como não lida;
- RF16 – O sistema deve apresentar na tela inicial do cliente as mensagens não lidas.
- RF17 – O sistema deve informar o estado atualizado de um contato durante uma conversação (*online* ou *offline*);
- RF18 – Ao receber uma nova mensagem, deve ser exibida uma tela de notificação;
- RF19 – O aplicativo de gerenciamento de cadastros deve possuir um modo de restaurar o estado de todos usuários para *offline;*
- RF20 – O aplicativo cliente deve permitir que o usuário envie e receba mensagens de texto;
- RF21 – Deve ser executado um aviso sonoro ao receber uma mensagem quando a janela de chat do contato estiver fechada, minimizada ou fora de foco.
- RF22 – Os usuários podem adicionar ou remover colaboradores de interesse à sua lista de contatos;
- RF23 – Os aplicativos cliente e gerenciamento de mensagens devem possuir uma tela de configuração de dados de acesso à base e ao serviço de controle de mensagens;

1.2 REQUISITOS NÃO FUNCIONAIS

- RNF01 – O sistema deve ser implementado na linguagem de programação Java 7;
- RNF02 – O sistema deve ser compatível com os sistemas operacionais Windows(XP ou superior) e Linux(Distribuições Ubuntu, Mint e Debian);
- RNF03 – O sistema deve utilizar banco de dados MySQL 5.6 ou superior;
- RNF04 – O sistema deve permitir que a base de dados seja mantida num servidor dedicado e/ou externo;
- RNF05 – Os sistema deve possuir um serviço de controle de mensagens(titulado "serviço" no Windows e "daemon" no Linux);

- RNF06 – O servidor e os clientes devem estar ligados através de uma estrutura de rede de comunicação TCP/IP;
- RNF07 – Todos as mensagens e dados de usuários como login e senha devem trafegar e ser mantidos de forma criptografada;
- RNF08 – O sistema deve permitir que a base de dados e serviço de controle operem em servidores separados e/ou em computação na nuvem;
- RNF09 – O serviço de controle de mensagens deve ser compatível com sistemas operacionais para servidores, como Ubuntu Server, Amazon Linux e Windows Server;
- RNF10 – Todas as configurações de acesso aos servidores ou portas de comunicação devem persistir em um arquivo de configuração(.cfg) na pasta pessoal de cada usuário.

1.3 NIVEIS DE ACESSO

- Administrador:

Mantém o cadastro de cargos, usuários, gerencia o serviço de mensagens e realiza todas ações disponíveis para usuários.

- Usuário:

Envia e recebe mensagens, consulta estado dos usuários, adiciona contatos à sua lista de amigos, verifica mensagens não lidas e acessa seu histórico de mensagens enviadas e recebidas.

2 ESPECIFICAÇÃO

2.1 CASOS DE USO

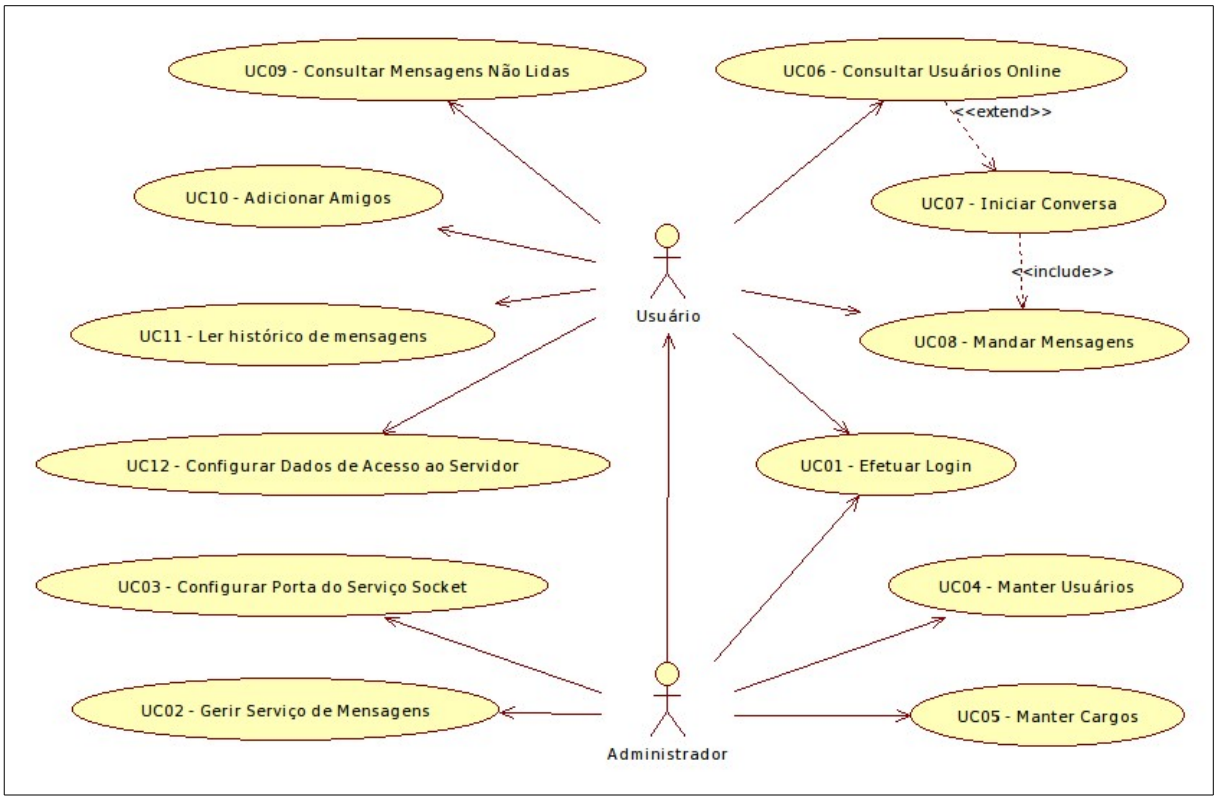

Imagem 1 – Diagrama de casos de uso.

- UC01 – Efetuar Login
  - Requisitos atendidos: RF05, RF10 e RF11.
  - Atores envolvidos: Administrador, Usuário.
  - Pré-condição: O funcionário deve estar cadastrado no sistema como usuário padrão ou administrador;
  - Cenário principal: O usuário informa login, senha e clica no botão entrar. O sistema exibe a tela inicial do aplicativo após confirmar conexão com o banco de dados e servidor de mensagem, verificar IP local, autenticar informações de acesso, verificar se usuário não está logado e salvar o novo IP na base de dados.
  - Cenário alternativo 01: Se não for possível conectar-se ao servidor, exibir a mensagem "Não foi possível realizar conexão com o Servidor!" e permanecer na tela de login.
  - Cenário alternativo 02: Se o login ou senha informados pelo usuário estiverem incorretos, exibir a mensagem "Usuário ou senha incorretos!" e permanecer na tela de login;
  - Cenário alternativo 03: Se usuário já estiver logado em outra máquina, exibir a mensagem "Usuário já logado!" e permanecer na tela de login;
  - Cenário alternativo 04: Quando o login for efetuado no aplicativo de controle de cadastros, deve ser verificado se usuário é administrador.

- UC02 – Gerir Serviço de Mensagens
  - Requisitos atendidos: RF09.
  - Atores envolvidos: Administrador.
  - Cenário principal: Iniciar ou parar o serviço(titulado "daemon" no Linux) de controle de mensagens;

- UC03 – Configurar Porta do Serviço *Socket*
  - Requisitos atendidos: RF09.
  - Atores envolvidos: Administrador.
  - Cenário principal: Na pasta pessoal do usuário do servidor, alterar o número da porta de conexão *Socket* no arquivo ".OpenLyncServico.cfg".

- UC04 -Manter Usuários
  - Requisitos atendidos: RF01, RF08, RF19.
  - Atores envolvidos: Administrador.
  - Cenário principal: Adicionar, editar ou remover usuários.
  - Cenário alternativo 01: Ao adicionar um usuário, se o login já estiver em uso exibir uma mensagem de aviso.
  - Cenário alternativo 02: Ao adicionar um usuário, se a foto de perfil não for informada pelo usuário utilizar a imagem padrão.
  - Cenário alternativo 03: Ao adicionar ou editar, se todos os campos obrigatórios (nome, login, senha, cargo) não forem preenchidos, exibir uma mensagem de aviso e impedir o cadastramento.

- UC05 – Manter Cargos
  - Requisitos atendidos: RF02, RF08.
  - Atores envolvidos: Administrador.
  - Cenário principal: Adicionar, editar ou remover cargos.

- Cenário alternativo 01: Ao adicionar ou editar, se o campo obrigatório (descrição do cargo) não for preenchido, exibir uma mensagem de aviso e impedir o cadastramento.
- Cenário alternativo 02: Ao remover, se o cargo estiver em uso por algum usuário, solicitar ao administrador se deseja alterar seus cargos para "nulo" e continuar a exclusão.

- UC06 – Consultar Usuários Online
  - Requisitos atendidos: RF14.
  - Atores envolvidos: Administrador, Usuário.
  - Cenário principal: Na tela inicial do aplicativo cliente são exibidos todos usuários com seus respectivos status.
  - Cenário alternativo 01: Ao marcar a caixa de seleção "Online" presente na tela principal, são exibidos apenas contatos com status *online*.

- UC07 – Iniciar Conversa
  - Requisitos atendidos: RF13, RF20.
  - Atores envolvidos: Administrador, Usuário.
  - Pré-condição: O usuário deve procurar o contato de interesse na lista geral ou de amigos.
  - Cenário principal: O usuário abre a tela de conversação clicando sobre o nome do contato na lista geral ou de amigos.
  - Cenário alternativo 01: Se uma conversa estiver em curso, o contato deve ser aberto em uma nova janela de chat.
  - Cenário alternativo 02: Se a janela de chat do contato clicado já estiver aberta, não realizar nenhuma ação.

- UC08 – Mandar e Receber Mensagens
  - Requisitos atendidos: RF07, RF15, RF17, RF18, RF20, RF21.
  - Atores envolvidos: Administrador, Usuário.
  - Cenário principal: O usuário envia ou recebe mensagens de texto na janela de chat respectiva ao contato destino/remetente.
  - Cenário alternativo 01: Se ao receber uma mensagem não houver uma janela de chat aberta, o sistema deve exibir uma mensagem de notificação e ativar um aviso sonoro. Quando esta for clicada, a janela de chat é aberta.
  - Cenário alternativo 02: Ao enviar uma mensagem, se o contato de destino estiver *offline,* o sistema deve marcar a mensagem como não lida.

- UC09 – Consultar Mensagens Não Lidas
  - Requisitos atendidos: RF16.
  - Atores envolvidos: Administrador, Usuário.
  - Cenário principal: Na tela inicial há um ícone azul onde ao clicar exibe uma lista com as mensagens não lidas (recebidas enquanto o usuário estava *offline*).
  - Cenário alternativo 01: Se não houver nenhuma mensagem não lida, o ícone deve se tornar cinza e ao clicar exibir a mensagem "Não há novas mensagens!".

- UC10 – Gerenciar Amigos
  - Requisitos atendidos: RF04, RF22.

- Atores envolvidos: Administrador, Usuário.
- Cenário principal: O usuário adiciona contatos de interesse à sua lista de amigos clicando no botão disponível na janela de chat.
- Cenário alternativo 01: Se o contato já estiver na lista de amigos, a ação do botão se torna “Remover amigo”.

- UC11- Ler Histórico de Mensagens
  - Requisitos atendidos: RF03, RF12.
  - Atores envolvidos: Administrador, Usuário.
  - Cenário principal: É exibido o histórico de todas conversas com um contato específico clicando no botão de histórico na janela de chat.

- UC12 – Configurar Dados de Acesso ao Servidor
  - Requisitos atendidos: RF23.
  - Atores envolvidos: Administrador, Usuário.
  - Cenário principal: Na tela de login do aplicativo cliente ou de gerenciamento de cargos, ao clicar no ícone em forma de engrenagem, exibir uma janela para configurar os dados de acesso aos servidores, como IP, usuário e senha do banco de dados, IP do serviço de mensagem e porta de conexão *socket.*

2.2 DIAGRAMA DE CLASSES

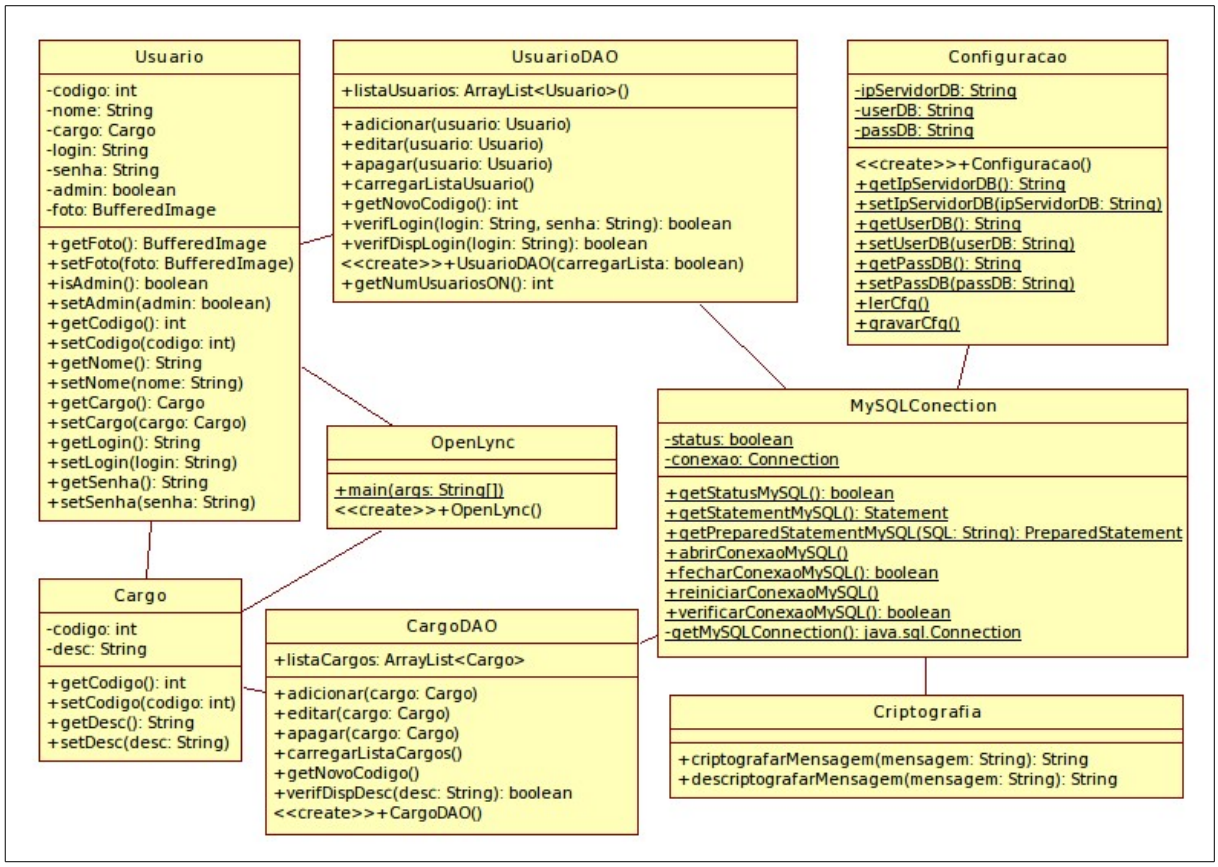

Imagem 2 – Diagrama de classes do aplicativo de gerenciamento de cadastros.

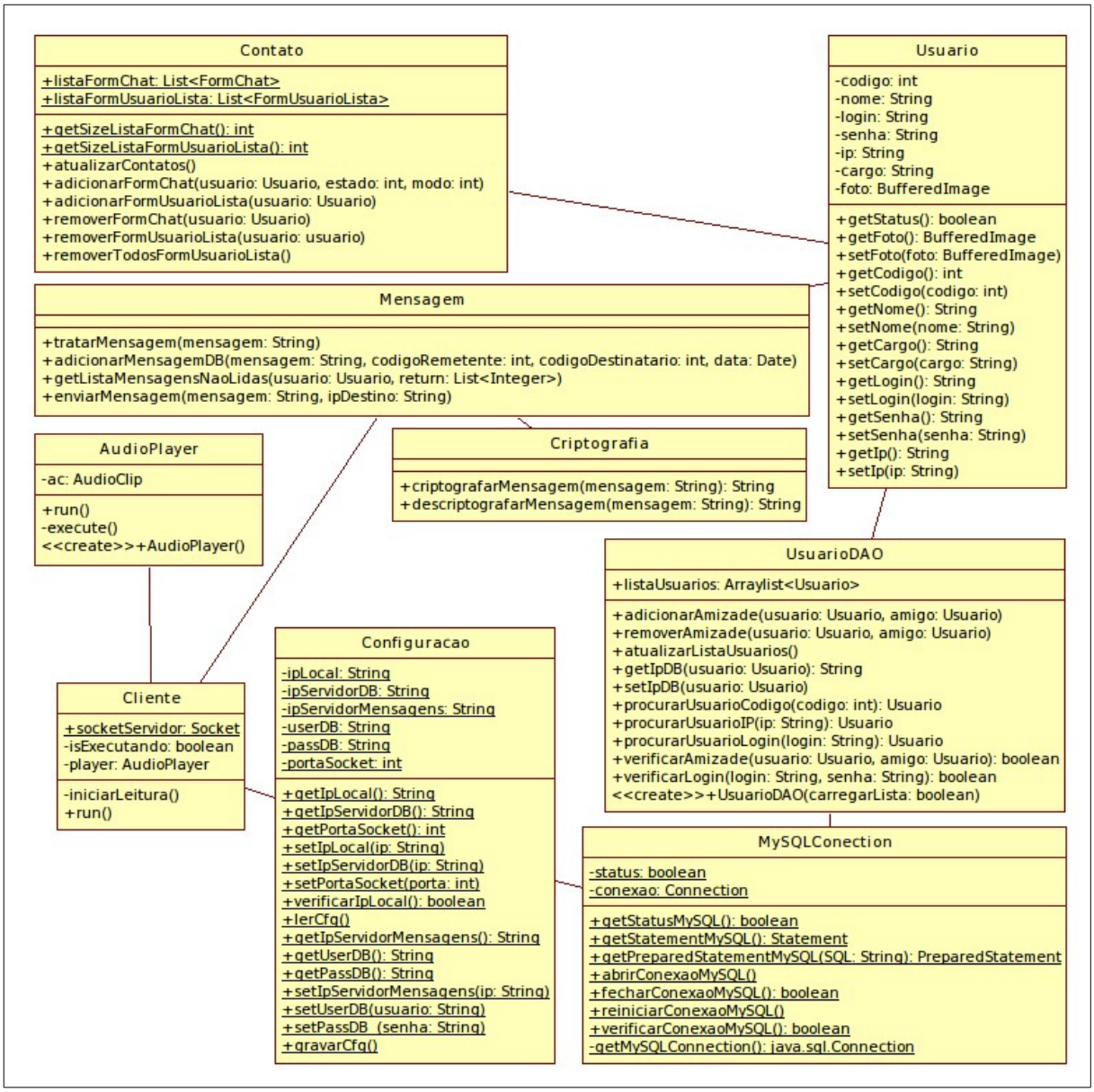

Imagem 3 -Diagrama de classes do aplicativo cliente.

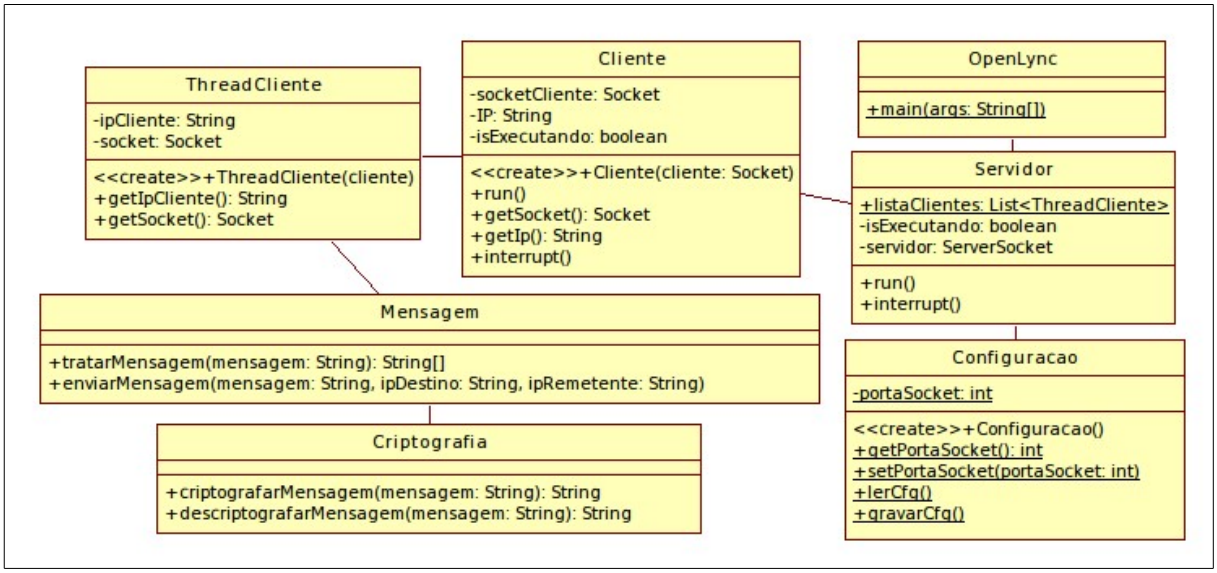

Imagem 4 -Diagrama de classes do serviço de controle de mensagens.

3 BANCO DE DADOS

3.1 MODELO CONCEITUAL

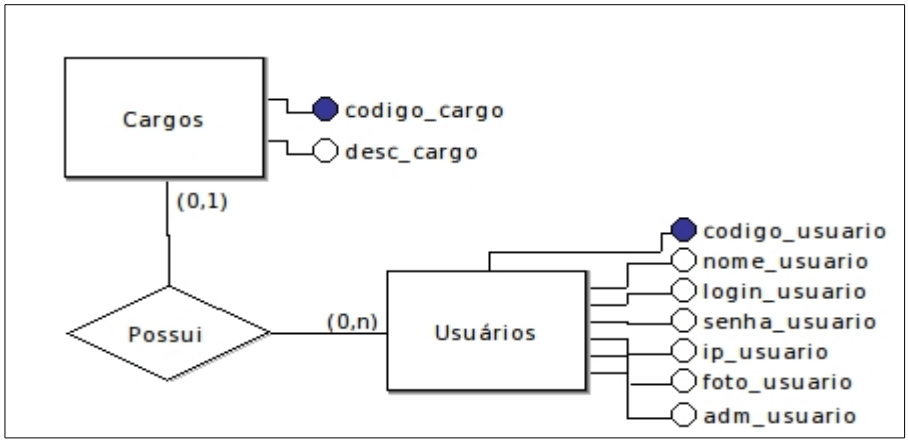

Imagem 5 – Modelo conceitual do banco de dados.

3.2 MODELO LÓGICO

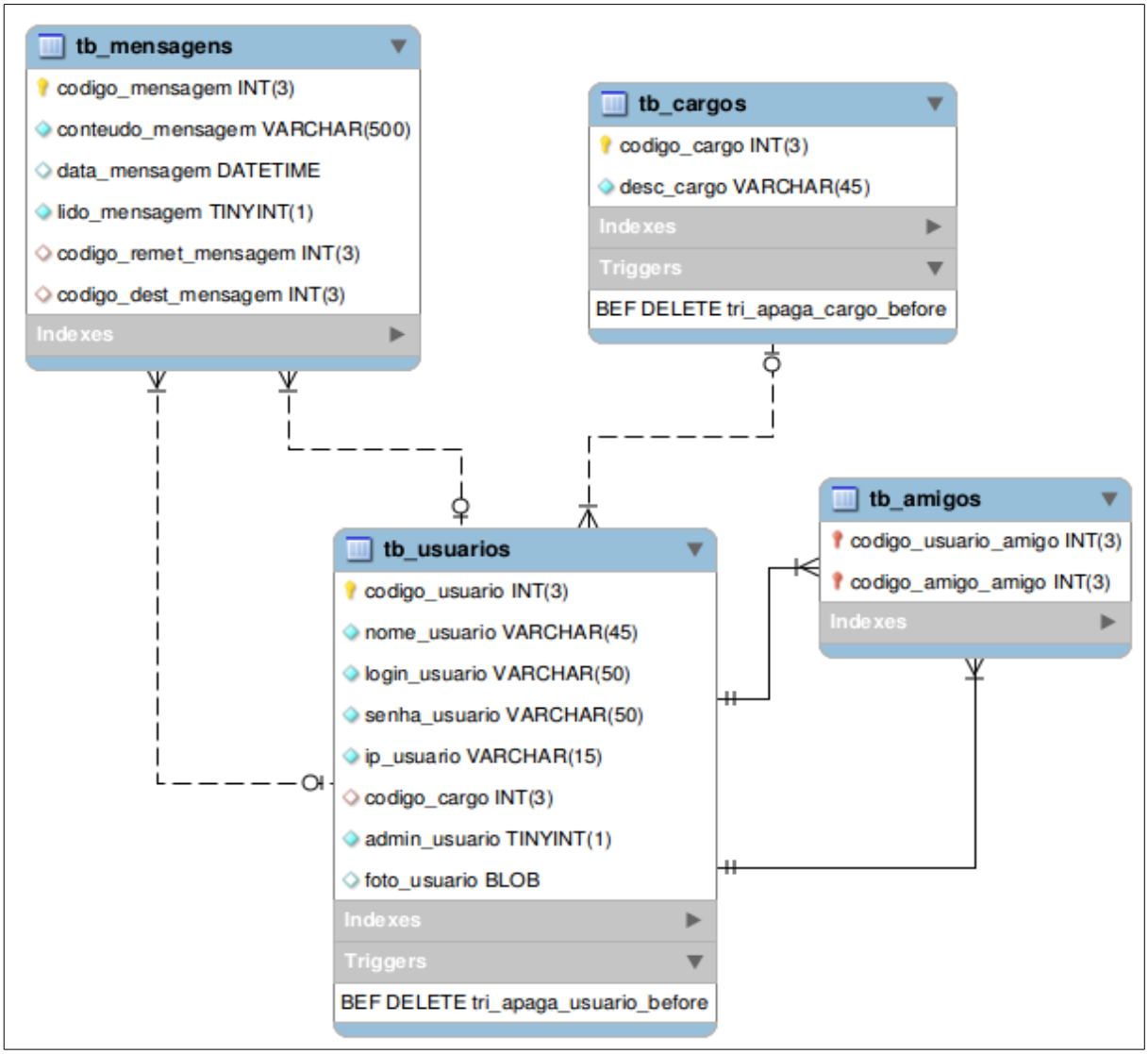

Imagem 6 – Modelo lógico do banco de dados.

3.3 SCRIPT SQL

```
CREATE DATABASE OpenLync;
USE OpenLync;

CREATE TABLE tb_cargos (
	codigo_cargo INT(3) PRIMARY KEY AUTO_INCREMENT,
	desc_cargo VARCHAR(45) NOT NULL
);

CREATE TABLE tb_usuarios (
	codigo_usuario INT(3) PRIMARY KEY AUTO_INCREMENT,
	nome_usuario VARCHAR(45) NOT NULL,
	login_usuario VARCHAR(50) NOT NULL,
	senha_usuario VARCHAR(50) NOT NULL,
	ip_usuario VARCHAR(15) NOT NULL,
	codigo_cargo INT(3),
	admin_usuario boolean NOT NULL,
	foto_usuario BLOB,

	CONSTRAINT codigo_cargo_fk FOREIGN KEY (codigo_cargo)
	REFERENCES tb_cargos(codigo_cargo)
);

CREATE TABLE tb_amigos (
	codigo_usuario_amigo INT(3),
	codigo_amigo_amigo INT(3),

	CONSTRAINT codigo_usuario_amigo_fk FOREIGN KEY (codigo_usuario_amigo)
	REFERENCES tb_usuarios(codigo_usuario),

	CONSTRAINT codigo_amigo_amigo_fk FOREIGN KEY (codigo_amigo_amigo)
	REFERENCES tb_usuarios(codigo_usuario),

	PRIMARY KEY (codigo_usuario_amigo, codigo_amigo_amigo)
);

CREATE TABLE tb_mensagens (
	codigo_mensagem INT(3) AUTO_INCREMENT,
	conteudo_mensagem VARCHAR(500) NOT NULL,
	data_mensagem DATETIME,
	lido_mensagem boolean NOT NULL,
	codigo_remet_mensagem INT(3),
	codigo_dest_mensagem INT(3),

	CONSTRAINT codigo_remet_mensagem_fk FOREIGN KEY
(codigo_remet_mensagem)
	REFERENCES tb_usuarios(codigo_usuario),

	CONSTRAINT codigo_dest_mensagem_fk FOREIGN KEY
(codigo_dest_mensagem)
	REFERENCES tb_usuarios(codigo_usuario),

	PRIMARY KEY (codigo_mensagem)
);
```

```
DELIMITER $$

/* Seleciona todas mensagens não lidas recebidas de um contato especifico */

CREATE PROCEDURE sp_getMensagensNaoLidas(IN codigo_usuario_login INT, IN
codigo_usuario_remetente INT)
BEGIN
	-- Carrega mensagens
	SELECT conteudo_mensagem, data_mensagem
	FROM tb_mensagens
	WHERE lido_mensagem = FALSE
		AND codigo_remet_mensagem = codigo_usuario_remetente
		AND codigo_dest_mensagem = codigo_usuario_login;

	-- Define mensagens como lidas
	UPDATE tb_mensagens
	SET lido_mensagem = TRUE
	WHERE codigo_remet_mensagem = codigo_usuario_remetente
		AND codigo_dest_mensagem = codigo_usuario_login;
END $$

/* Seleciona todas mensagens enviadas e recebidas de um contato especifico */

CREATE PROCEDURE sp_getHistoricoMensagens(IN codigo_usuario_login INT, IN
codigo_usuario_remetente INT)
BEGIN
	-- Carrega mensagens
	SELECT conteudo_mensagem, data_mensagem, codigo_remet_mensagem
	FROM tb_mensagens
	WHERE codigo_remet_mensagem = codigo_usuario_remetente
		OR codigo_remet_mensagem = codigo_usuario_login
		AND codigo_dest_mensagem = codigo_usuario_login
		OR codigo_dest_mensagem = codigo_usuario_remetente;

	-- Define mensagens como lidas
	UPDATE tb_mensagens
	SET lido_mensagem = TRUE
	WHERE codigo_remet_mensagem = codigo_usuario_remetente
		OR codigo_remet_mensagem = codigo_usuario_login
		AND codigo_dest_mensagem = codigo_usuario_login
		OR codigo_dest_mensagem = codigo_usuario_remetente;
END $$

/* Adiciona uma nova mensagem. Se o usuário destino estiver offline, a mensagem é
marcada como não lida */

CREATE PROCEDURE sp_adicionarMensagem(IN codigo_remetente INT, IN
codigo_destinatario INT, IN mensagem VARCHAR(500), IN data DATETIME)
BEGIN
	SELECT ip_usuario INTO @ipDest
	FROM tb_usuarios
	WHERE codigo_usuario = codigo_destinatario;

	IF (@ipDest = 'null') THEN
		SET @lidoMensagem = FALSE;
	ELSE
		SET @lidoMensagem = TRUE;
```

```
        END IF;

        INSERT INTO tb_mensagens(
                conteudo_mensagem,
                data_mensagem,
                lido_mensagem,
                codigo_remet_mensagem,
                codigo_dest_mensagem)
        VALUES (mensagem,
                        data,
                        @lidoMensagem,
                        codigo_remetente,
                        codigo_destinatario);
END $$

/* Retorna Verdadeiro se o contato informado estiver na lista de amigos */

CREATE FUNCTION fc_verificarAmizade(codigo_usuario INT, codigo_amigo INT)
RETURNS BOOLEAN
BEGIN
        IF (SELECT 1 FROM tb_amigos
                WHERE codigo_usuario_amigo = codigo_usuario
                        AND codigo_amigo_amigo = codigo_amigo) THEN
                RETURN TRUE;
        ELSE
                RETURN FALSE;
        END IF;
END $$

/* Cria view substituindo campo 'codigo_cargo' pela descrição do mesmo */

CREATE OR REPLACE VIEW vw_usuarios AS
        SELECT codigo_usuario,
                nome_usuario,
                login_usuario,
                senha_usuario,
                foto_usuario,
                ip_usuario,
                admin_usuario,
                tb_cargos.desc_cargo
        FROM tb_usuarios
                LEFT JOIN tb_cargos
                ON tb_usuarios.codigo_cargo = tb_cargos.codigo_cargo;

/* Altera o cargo para 'nulo' de todos usuários cadastrados com o registro a ser apagado
*/

CREATE TRIGGER tri_apaga_cargo_before BEFORE DELETE
ON tb_cargos FOR EACH ROW
BEGIN
        UPDATE tb_usuarios
        SET codigo_cargo = NULL
        WHERE codigo_cargo = OLD.codigo_cargo;
END $$

/* Apaga histórico de mensagens e amigos do usuário que será excluido */
```

```
CREATE TRIGGER tri_apaga_usuario_before BEFORE DELETE
ON tb_usuarios FOR EACH ROW
BEGIN
        DELETE FROM tb_amigos
        WHERE codigo_usuario_amigo = OLD.codigo_usuario
                OR codigo_amigo_amigo = OLD.codigo_usuario;

        DELETE FROM tb_mensagens
        WHERE codigo_remet_mensagem = OLD.codigo_usuario
                OR codigo_dest_mensagem = OLD.codigo_usuario;
END $$

[...]
```

Quadro 1 – Algoritmo SQL para criação da base de dados, procedimentos, funções e *triggers*.

4 IMPLEMENTAÇÃO

4.1 TELAS

Imagem 7 – Janela de contatos.

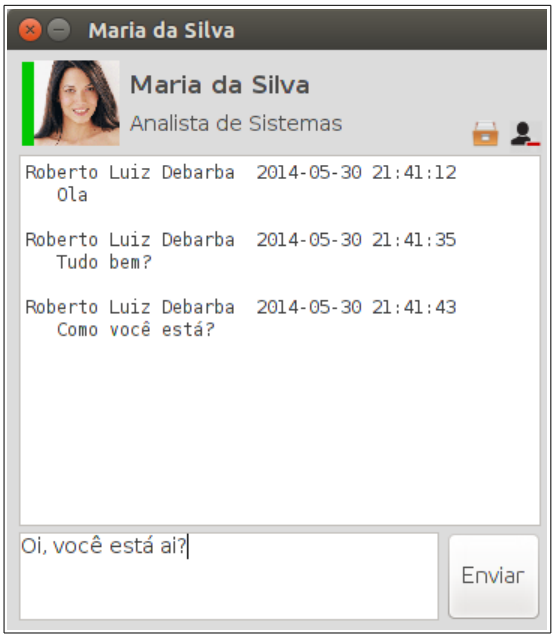

Imagem 8 – Janela de chat.

Imagem 9 – Janela principal do aplicativo de gerenciamento de cargos.

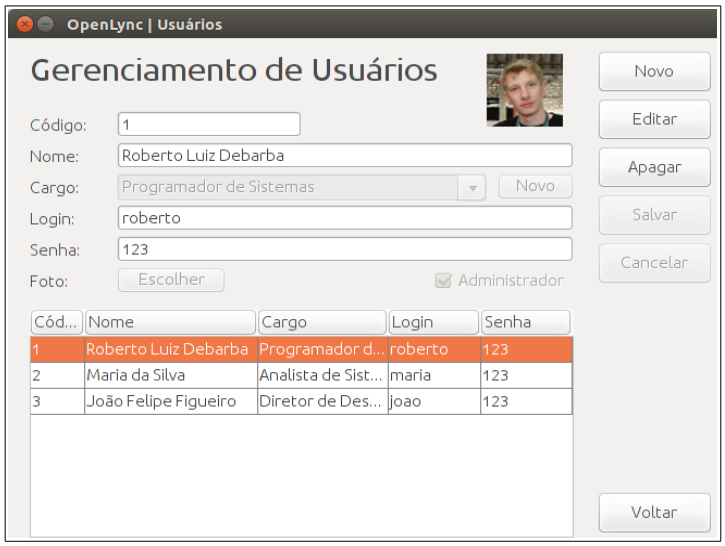

Imagem 10 – Janela de manutenção de usuários.

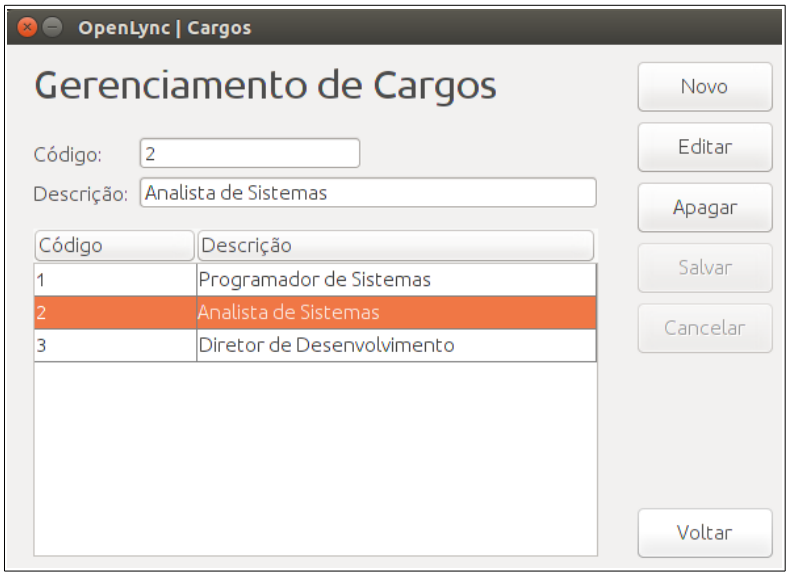

Imagem 11 – Janela de manutenção de cargos.

4.2 DESENVOLVIMENTO DO CÓDIGO FONTE

```
public void run() {
	try {
		isExecutando = true;
		servidor = new ServerSocket(Configuracao.getPortaSocket());

		while (isExecutando) {

			Socket socketCliente = null;
			try {
				socketCliente = servidor.accept();
				System.out.println("Nova conexão com o cliente
				    "+socketCliente.getInetAddress().getHostAddress());

				ThreadCliente cliente = new ThreadCliente(new
				    Cliente(socketCliente));

				listaClientes.add(cliente);
				cliente.start();

			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	} catch (IOException e1) {
		e1.printStackTrace();
	}
}
```

Quadro 2 – Serviço de mensagens. Inicialização do servidor *Socket.*

```
public void atualizarContatos() {

	UsuarioDAO dao = new UsuarioDAO(true);

	/*
	 * Varre listaFormUsuarioLista e dao.ListaUsuarios comparando os valores
	 * Se encontrou atualiza o status
	 * Se nao encontrou cria novo FormUsuarioLista
	 */

	//Varre todos Usuarios para comparar com InternalFrames
	for (int i = 0; i < dao.listaUsuarios.size(); i++) {

		//Se usuario for o que efetuou login = ignora
		if (!dao.listaUsuarios.get(i).getLogin().equals(FormLogin
		    .getUsuarioLogin().getLogin())) {

			//Varre InternalFrames ate achar o que corresponde ao usuario
			boolean achou = false;
			int z = 0;
			while (z < listaFormUsuarioLista.size()) {
				if (listaFormUsuarioLista.get(z).getUsuario().getCodigo()
				    == dao.listaUsuarios.get(i).getCodigo()) {
					achou = true;
					break;
```

```
                        }
                        z++;
                    }

                    //Se achou
                    if (achou) {
                        //Atualiza status
                        listaFormUsuarioLista.get(z).setStatus(dao.listaUsuarios
                            .get(i).getStatus(), dao.listaUsuarios.get(i).getIp());
                        //Se não achou
                    } else {
                        // Verificação de Abas e opção Online ---
                        boolean adicionarIF = false;

                        //Aba "Todos"
                        if (FormIncial.indiceAba == 0) {
                            // Considera opção "Apenas Online" do FormIncial
                            if (FormIncial.checkOnline.isSelected()) {
                                //Se estiver online = adiciona
                                if (dao.listaUsuarios.get(i).getStatus()) {
                                    adicionarIF = true;
                                }
                            } else {
                                adicionarIF = true;
                            }
                        //Aba "Amigos"
                        } else if (FormIncial.indiceAba == 1) {
                            // Considera opção "Apenas Online" do FormIncial
                            if (FormIncial.checkOnline.isSelected()) {
                                //Se estiver online = adiciona
                                if (dao.listaUsuarios.get(i).getStatus()) {
                                    //Se for amigo
                                    if (dao.verificarAmizade(FormLogin
                                        .getUsuarioLogin(),
                                            dao.listaUsuarios.get(i))) {
                                         adicionarIF = true;
                                    }
                                }
                            } else {
                                //Se for amigo
                                if (dao.verificarAmizade(FormLogin
                                   .getUsuarioLogin(),
                                       dao.listaUsuarios.get(i))) {
                                      adicionarIF = true;
                                }
                            }
                        }

                        if (adicionarIF) {
                            adicionarFormUsuarioLista(dao.listaUsuarios.get(i));
                        }
                    }
                }
            }
        }
}
```

Quadro 3 – Aplicativo cliente. Atualização da lista de contatos da tela principal.

```
public void adicionar(Usuario usuario) {

	Criptografia cript = new Criptografia();
	try {
		String SQL = "CALL sp_adicionarUsuario(?, ?, ?, ?, ?, ?, ?, ?)";

		PreparedStatement pst = MySQLConection
		    .getPreparedStatementMySQL(SQL);

		pst.setInt(1, usuario.getCodigo());
		pst.setString(2, usuario.getNome());
		pst.setString(3, usuario.getCargo());
		pst.setString(4, cript.criptografarMensagem(usuario.getLogin()));
		pst.setString(5, cript.criptografarMensagem(usuario.getSenha()));
		pst.setString(6, "null");
		pst.setBoolean(7, usuario.isAdmin());

		// Prepara imagem para INSERT
		if (usuario.getFoto() != null) {
			ByteArrayOutputStream out = new ByteArrayOutputStream();
			try {
				ImageIO.write(usuario.getFoto(), "jpeg", out);
			} catch (IOException e) {
				e.printStackTrace();
			}

			byte[] buf = out.toByteArray();
			ByteArrayInputStream inStream = new
			    ByteArrayInputStream(buf);

			pst.setBinaryStream(8, inStream, inStream.available());
		} else {
			pst.setBinaryStream(8, null);
		}

		pst.executeUpdate();
	} catch (SQLException e) {
		e.printStackTrace();
	}
}
```

Quadro 4 – Aplicativo de controle de cadastros. Adição de novo usuário ao banco de dados.

5 REFERÊNCIAS

ALEXANDER, Alvin. **A Java MySQL SELECT:** Example. 2013. Disponível em: <http://alvinalexander.com/java/java-mysql-select-query-example>. Acesso em: 20 maio 2014.

BIANCHI, Wagner. **MySQL – TRIGGERS**. Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/mysql-triggers/8088>. Acesso em: 20 maio 2014.

CAELUM. **Apostila Java e Orientação a Objetos:** Capítulo 17, Programação Concorrente e Threads. Disponível em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/programacao-concorrente-e-threads/>. Acesso em: 20 maio 2014.

CAELUM. **Apostila Java e Orientação a Objetos**: Capítulo 19, Apêndice - Sockets. Disponível em: <http://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/>. Acesso em: 20 maio 2014.

DEVMEDIA, Equipe. **Procedures e funções no MySQL**. Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/procedures-e-funcoes-no-mysql/2550>. Acesso em: 20 maio 2014.

FREIXO, Bruno. **Criando um cadastro de usuário em Java.** 2012. Publicado por Oficina da Net. Disponível em: <http://www.oficinadanet.com.br/artigo/java/criando-um-cadastro-de-usuario-em-java>. Acesso em: 20 maio 2014.

JAVA2S. **Dynamic menu item for MDI children window and scroll bar**. Publicado por Java2s. Disponível em: <http://www.java2s.com/Code/Java/Swing-JFC/DynamicmenuitemforMDIchildrenwindowandscrollbar.htm>. Acesso em: 20 maio 2014.

LANHELLAS, Ronaldo. **Trabalhando com Threads em Java.** Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/trabalhando-com-threads-em-java/28780>. Acesso em: 20 maio 2014.

MAZZI, Carlos Eduardo Domingues. **Criando uma conexão Java + MySQL Server.** Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/criando-uma-conexao-java-mysql-server/16753>. Acesso em: 20 maio 2014.

ORACLE. **How to Use Internal Frames**. Disponível em: <http://docs.oracle.com/javase/tutorial/uiswing/components/internalframe.html>. Acesso em: 20 maio 2014.

PEREIRA, Paulo Sérgio. **Sockets com Java**. Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/sockets-com-java-parte-i/9465>. Acesso em: 20 maio 2014.

PÉREZ, Ivan Mora. **Deploying Java Unix Daemon with Java Service Wrapper.** 2013. Disponível em: <http://opentodo.net/2013/03/deploying-java-unix-daemon-with-java-service-wrapper/>. Acesso em: 20 maio 2014.

RODRIGUES, Joel. **Stored Procedures no MySQL**. Publicado por DevMedia. Disponível em: <http://www.devmedia.com.br/stored-procedures-no-mysql/29030>. Acesso em: 20 maio 2014.

VOGEL, Lars. **Java concurrency (multi-threading):** Tutorial. 2013. V2.3. Disponível em: <http://www.vogella.com/tutorials/JavaConcurrency/article.html>. Acesso em: 20 abr. 2014.