GingaFrEvo & GingaRAP

Evolução do Middleware Ginga para Múltiplas Plataformas (Componentização)
&
Ferramentas para Desenvolvimento e Distribuição de Aplicações Declarativas

Relatório Técnico RT- 24
Manual Operacional do Componente P2P (*peer-to-peer*)

PROGRAMA CTIC
REDE NACIONAL DE ENSINO E PESQUISA

Apoio MCT

HISTÓRICO

Data	Versão	Descrição	Autor
29/01/10	0.1	Elaboração Inicial do Documento	Diogo Pedrosa e José Augusto Martins
14/06/10	0.2	Ajustes	Diogo Pedrosa
30/06/10	0.3	Ajustes na instalação da libJingle e na descrição da aplicação de teste	Diogo Pedrosa e José Augusto Martins
17/11/10	0.9	Acréscimo do suporte a recebimento de arquivos, a chat e mudança de status e recebimento de notificação de mudança de status.	Diogo Pedrosa
18/11/10	1.0	Revisão final do documento	Erick Melo

ÍNDICE

1 Introdução

Este componente tem por objetivo proporcionar a colaboração entre usuários via servidores apropriados.

A demanda pela padronização do *middleware* Ginga para plataformas de IPTV, Internet TV e P2P TV abre inúmeras possibilidades para que usuários dessas plataformas realizem atividades colaborativas de forma síncrona ou assíncrona. O compartilhamento de anotações, por exemplo, já foi demonstrado no protótipo da ferramenta **WaC** (*Watch and Comment*), projetada para permitir que usuários sem treinamento possam realizar e compartilhar anotações e criar vídeos interativos NCL. Esse tipo de colaboração, já comum entre usuários Web, é uma importante alternativa a ser explorada com a introdução do Ginga no mundo Internet.

O protocolo XMPP (Protocolo Extensível de Mensagem e Presença) é uma tecnologia aberta para comunicação em tempo-real que permite incluir mensagens instantâneas, presença, conferências em chats, ligações de voz e vídeo, colaboração, e roteamento generalizado de dados XML em uma ampla gama de aplicações[1].

Como o próprio nome diz, o protocolo XMPP permite a criação de extensões para determinados propósitos. Entre as extensões criadas, a extensão JINGLE (XEP-0166) e seus complementos (XEP-0167, XEP-0176, XEP-0177 e XEP-0181) apareceram de forma promissora com o objetivo de iniciar e gerenciar sessões p2p multimídia entre duas entidades XMPP de maneira que seja interoperável com padrões existentes da internet[2].

Para a criação deste componente ficou definida a utilização tanto do protocolo XMPP como da extensão JINGLE. Para promover o reuso, ajudar com o desenvolvimento e também usufruir das melhorias feitas no protocolo, uma biblioteca chamada **libJingle**[3] foi escolhida. Essa biblioteca foi projetada pelo Google® e fornece a lógica necessária para se projetar aplicações utilizando os protocolos adotados. Ela está em fase de desenvolvimento, mas já está sendo utilizada pelo comunicador **Google Talk**[4].

1.1 Objetivos

Este documento descreve o manual operacional de uso do Componente P2P. É descrita a API de uso do componente além da estruturação do código-fonte e instruções para compilação, instalação e utilização.

1.2 Escopo

Este documento está associado ao projeto GingaFrEvo. O projeto é divido em diversos subprojetos. O componente especificado neste documento enquadra-se no subprojeto Ginga-Aiyê, atendendo aos objetivos de permitir o desenvolvimento de aplicações que permitem a colaboração entre usuários através do paradigma de comunicação P2P.

---

1 Para maiores informações dirija-se a www.xmpp.org. Última visita maio/2010

2 Traduzido livremente de http://xmpp.org/extensions/xep-0166.html. Última visita maio/2010

3 Para maiores informações dirija-se a http://code.google.com/intl/pt-BR/apis/talk/libjingle/index.html. Última visita maio/2010

4 http://www.google.com/talk/intl/pt-BR/about.html

1.3 Estrutura do Documento

O documento está estruturado da seguinte forma: a Seção 2 apresentado um tutorial de instalação do componente; a Seção 3 apresenta detalhes técnicos do componente: diagrama de classes e descrição da API do componente; a Seção 3 ainda apresenta exemplos de uso do componente.

2 Instalação

A instalação do componente envolve também a instalação da biblioteca **libJingle**. Além disso, foi desenvolvida também uma aplicação para testar o componente. A chamada da aplicação é feita através da main do Ginga.

2.1 Conteúdo do componente

O Componente P2P possui arquivos de cabeçalho e de código fonte C++ e arquivos de projeto para compilação do módulo. Os arquivos estão localizados nos seguintes diretórios:

gingacc-p2p (componente peer-to-peer)	
Tipo de arquivo	Diretório
Arquivos de cabeçalho	gingacc-p2p/include/
Arquivos de código fonte	gingacc-p2p/src/
Arquivos de projeto	gingacc-p2p

P2PtestApp (aplicação de teste do componente)	
Tipo de arquivo	Diretório
Arquivos de cabeçalho	p2p-test-app/include/
Arquivos de código fonte	p2p-test-app/src/
Arquivos de projeto	p2p-test-app/

Libjingle (biblioteca do protocolo Jingle)	
Tipo de arquivo	Diretório
Arquivos de cabeçalho	libjingle/talk/base/include/ libjingle/talk/xmllite/ libjingle/talk/xmpp/ libjingle/talk/p2p/base/ libjingle/talk/p2p/client/ libjingle/talk/session/fileshare/ libjingle/talk/session/phone/ libjingle/talk/session/tunnel/ libjingle/talk/thid_party/gips/ libjingle/talk/thid_party/mediastreamer/ libjingle/talk/examples
Arquivos de código fonte	libjingle/talk/base/src/ libjingle/talk/xmllite/ libjingle/talk/xmpp/ libjingle/talk/p2p/base/ libjingle/talk/p2p/client/ libjingle/talk/session/fileshare/ libjingle/talk/session/phone/ libjingle/talk/session/tunnel/ libjingle/talk/thid_party/gips/ libjingle/talk/thid_party/mediastreamer/ libjingle/talk/examples
Arquivos de projeto	libjingle/

2.2 Requisitos do Sistema

O Componente P2P requer:

Software	URL
gingacc-system	http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/
libjingle*	http://lince.dc.ufscar.br/redmine/projects/ginga-ctic

*Apesar da libjingle ser distribuída pelo Google, por determinação do projeto ela gera originalmente bibliotecas estáticas, que devem ser adicionadas por aplicativos e assim ficarem auto-contidas dentro do mesmo. No caso da utilização pelo Ginga isso não é muito recomendado, pois cada aplicação que quiser fazer uso de tal biblioteca iria incluir a mesma dentro de seu executável gerando assim aplicações com tamanhos consideravelmente grandes. Em contrapartida, com a geração de bibliotecas compartilhadas todas as aplicações podem se beneficiar de tais bibliotecas em um diretório centralizado. Assim modificações nos arquivos makefile.am foram feitas no projeto da libJingle para que ela gerasse bibliotecas compartilhadas.

2.3 Compilação e Instalação

O código fonte do Componente P2P e suas dependências pode ser obtido no seguinte endereço:

URL
http://lince.dc.ufscar.br/redmine/projects/ginga-ctic

2.3.1 LigJingle

A libjingle pode ser compilada e instalada através dos seguintes passos:

```
cd libjingle/
./autogen.sh
make
sudo make install
```

A biblioteca também precisa ser incluida no path do sistema:

```
ldconfig /usr/local/lib/libjingle
```

2.3.2 Componente P2P

O componente P2P pode ser compilado e instalado através dos seguintes passos:

```
cd gingacc-cpp/gingacc-p2p
./autogen.sh
make
sudo make install
```

2.3.3 Teste

A aplicação de teste pode ser compilada e instalada através dos seguintes passos:

```
cd gingacc-cpp/p2p-test-app
./autogen.sh
make
sudo make install
```

3 Componente P2P

Este componente permite a troca de mensagens de texto (chat) e o envio e recebimento de arquivos de qualquer tipo por parte de aplicações residentes em C++ que executam sobre o Ginga. A aplicação com a qual a comunicação é feita precisa ser uma aplicação Ginga ou um cliente XMPP. Pelo fato de a libjingle ainda não está finalizada, a troca de arquivo ainda só é suportada por clientes que também utilizam a libjingle. Os clientes usados nos testes de troca de arquivo foram o pcp (uma aplicação de exemplo que é disponibilizada junto com a libJingle), o gtalX[5] e o Google Talk. Os clientes usados nos testes de chat foram o gtalX, o Google Talk, o Empathy[6] e o Pidgin[7].

A Seção 3.1 apresenta as principais funcionalidades do componente. A Seção 3.2 apresenta o diagrama de classes da solução e a Seção 3.3 explica como uma aplicação pode fazer uso de cada uma de suas funcionalidades.

3.1 Principais Funcionalidades

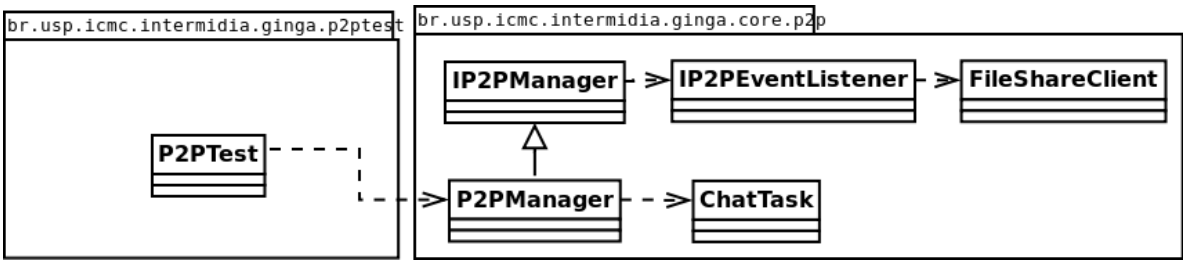
As principais funcionalidades providas pelo componente P2P são:

- Conexão em um servidor XMPP
- Desconexão do servidor
- Envio de mensagem de bate-papo
- Recebimento de mensagem de bate-papo
- Envio de arquivos e diretórios
- Recebimento de arquivos e diretórios
- Definição do status do usuário
- Recebimento de alteração do status de amigos

3.2 Diagrama de Classes

A figura abaixo apresenta o diagrama de classes que mostra tanto as classes do Componente P2P, quanto a classe da aplicação de teste do componente (P2PTestApp).

[5] http://sites.google.com/site/jozsefbekes/Home/gtalx
[6] http://live.gnome.org/Empathy
[7] http://www.pidgin.im/



3.3 Utilização

O componente foi desenvolvido utilizando a versão 0.11.2 da implementação de referência.

Para testar o componente, foi criada uma aplicação em forma de biblioteca .so. A aplicação deve herdar da classe ::br::pucrio::telemidia::ginga::core::system:: thread::Thread, para que o Ginga não fique travado enquanto espera a execução da mesma.

A chamada da aplicação foi realizada no arquivo main.cpp do projeto ginga-cpp, como mostrado no trecho de código abaixo.

```
#include "p2ptestapp/P2PTest.h"
using namespace ::br::usp::icmc::intermidia::ginga::p2ptest;
...
        bool enableMultimodalTestApp = false;
        bool enableP2PTestApp = false;
...
        // Acrescentar no fim do método printHelp.
        cout << "    --enable-p2p-test-app Enable the execution of the application";
        cout << " that tests the gingacc-p2p component." << endl;
...
        // Acrescentar entre os códigos que testem por parâmetros.
        } else if (strcmp(argv[i], "--enable-p2p-test-app") == 0) {
                enableP2PTestApp = true;
...
        // Acrescentar no else que indica que o devClass é igual a 0.
        if (enableP2PTestApp) {
                P2PTest *p2pTest = P2PTest::getInstance();
        }
```

Uma aplicação que queira fazer uso do componente deve acessar a instância única do P2PManager através do método ***getInstance()*** como mostrado no exemplo de código abaixo, retirado da classe P2PTest.

```
#include <talk/base/cryptstring.h>
#include <p2p/P2PManager.h>
using namespace ::br::usp::icmc::intermidia::ginga::core::p2p;
...
P2PManager *p2p = P2PManager::getInstance();
```

As seguintes seções exemplificam o uso de cada uma das funcionalidades listadas na Seção 3.1. Para mais informações, consulte a documentação de código do componente, disponibilizada junto com o mesmo.

3.3.1 Conexão em um servidor XMPP

Para se conectar a um servidor XMPP, a aplicação deve fornecer url do servidor, porta, nome de usuário e senha, os quais podem ser obtidos do usuário via interface gráfica, ou consultados em um arquivo de configuração previamente salvo pela aplicação, como mostrado no exemplo abaixo.

```
ifstream p2pConfigFile("/usr/local/etc/ginga/files/p2p.conf");

if (p2pConfigFile.is_open()) {
        string server;
        string port;
        string username;

        getline(p2pConfigFile, server);
        getline(p2pConfigFile, port);
        getline(p2pConfigFile, username);

        talk_base::InsecureCryptStringImpl pass;
        std::string& passStr = pass.password();
        getline(p2pConfigFile, passStr);

        p2pConfigFile.close();

        p2p->connect(server, atoi(port.c_str()), username, pass, this);
} else {
        LoggerUtil_info(logger, "Não foi possível abrir arquivo de"
                        "configuração");
}
```

Quando a conexão for estabelecida, a aplicação receberá uma notificação através do método ***stateChange()***, da interface IP2PEventListener. O quinto parâmetro da chamada do método ***connect()*** indica qual objeto IP2PEventListener será responsável por tratar esse tipo de mensagem e outras explicadas nas próximas seções. O trecho de código abaixo mostra um possível tratamento para mensagens de mudança de estado de conexão.

```
void P2PTest::stateChanged(buzz::XmppEngine::State state) {

        P2PTest* p2pTest = P2PTest::getInstance();

        p2pTest->state = state;



        // O P2PManager notifica usando STATE_NONE quando a mudança não é no estado

        // e sim no status do próprio usuário.

        if (state == buzz::XmppEngine::STATE_OPEN ||

                        state == buzz::XmppEngine::STATE_CLOSED ||

                        state == buzz::XmppEngine::STATE_NONE) {

                p2pTest->showMainScreen();

        }


}
```

3.3.2 Desconexão do servidor

Para se desconectar, basta chamar o método ***disconnect()***, como mostrado abaixo.

```
p2p->disconnect();
```

A aplicação será novamente notificada da mudança de estado da conexão, como explicado na seção anterior.

3.3.3 Envio de mensagem de bate-papo

O envio de mensagens de texto para um amigo, é feito através do método ***sendChatMessage()***, como mostrado abaixo.

```
p2p->sendChatMessage(destinationStatus->jid().BareJid().Str(), "Oi!");
```

Como pode ser percebido, a identificação do destinatário nesse exemplo se dá através do endereço XMPP (*Jabber Identifier*) sem a parte relativa ao recurso. Ou seja, sendo o endereço completo node@domain/resource, o endereço usado será node@domain, pois assim o destinatário receberá a mensagem em todos os clientes em que estiver conectado.

3.3.4 Recebimento de mensagem de bate-papo

O recebimento de mensagens de bate-papo se dá através do método ***chatMessageReceived()*** da interface IP2PEventListener, como mostrado no exemplo abaixo.

```
void P2PTest::chatMessageReceived(const string& from, const string& text) {

        P2PTest* p2pTest = P2PTest::getInstance();



        // Mostra mensagem recebida para o usuário
        ...
}
```

3.3.5 Envio de arquivos e diretórios

O envio de arquivos e diretórios para um amigo, é feito através do método ***sendFile()***, como mostrado abaixo.

```
p2p->sendFile(fileOrFolderToTransfer, destinationJid);
```

3.3.6 Recebimento de arquivos e diretórios

O recebimento de arquivos e diretórios se dá através do método ***transferStateChanged()*** da interface IP2PEventListener. Esse método será chamado várias vezes, desde o oferecimento do item, passando por atualizações do status da transferência, até a finalização da transferência, como mostrado no exemplo abaixo.

```
void P2PTest::transferStateChanged(FileShareClient::State state,
                const cricket::FileShareManifest* manifest) {

        P2PTest* p2pTest = P2PTest::getInstance();

        switch (state) {
```

```
        case FileShareClient::OFFER: {
                stringstream manifestDescription;

                if (manifest->size() == 1) {
                        manifestDescription <<  manifest->item(0).name;
                        p2pTest->fileTransfered = manifest->item(0).name;
                } else if (manifest->GetFileCount() && manifest->GetFolderCount()) {
                        manifestDescription <<  manifest->GetFileCount() <<
                                        " arquivos e " << manifest->GetFolderCount() <<
                                        " pastas";
                } else if (manifest->GetFileCount() > 0) {
                        manifestDescription <<  manifest->GetFileCount() << " arquivos";
                } else {
                        manifestDescription <<  manifest->GetFolderCount() << " pastas";
                }

                p2pTest->showOffer(manifestDescription.str());
                break;
        }
        case FileShareClient::TRANSFER: {
                // Permite mostrar barra de progresso.
                break;
        }
        case FileShareClient::COMPLETE: {
                p2pTest->showSuccess();
                break;
        }
        case FileShareClient::LOCAL_CANCEL:
        case FileShareClient::REMOTE_CANCEL:
        case FileShareClient::FAILURE:{
                // Permite mostrar mensagem de erro.
                break;
        }
        }
}
```

3.3.7 Definição do status do usuário

A definição do status atual do usuário é feita através do método ***setStatus()***. Ele permite que se defina um do cinco estados previstos pelo protocolo XMPP e ainda uma mensagem personalizada, como mostrado abaixo.

```
        if (code >= CodeMap::KEY_1 && code <= CodeMap::KEY_5) {
                IP2PManager::Status s;

                if (code == CodeMap::KEY_1) {
                        s = IP2PManager::CHAT;
                } else if (code == CodeMap::KEY_2) {
                        s  = IP2PManager::ONLINE;
                } else if (code == CodeMap::KEY_3) {
                        s = IP2PManager::DND;
                } else if (code == CodeMap::KEY_4) {
                        s = IP2PManager::AWAY;
                } else if (code == CodeMap::KEY_5) {
                        s = IP2PManager::XA;
                }

                p2p->setStatus(s, “Feliz!” );
        }
```

3.3.8 Recebimento de alteração do status de amigos

Por fim, a aplicação será notificada sobre mudanças de status dos amigos do usuário conectado através do método ***friendsStatusChanged()***, da interface IP2PEventListener, como mostrado abaixo.

```
void P2PTest::friendsStatusChanged(const buzz::Status &status) {
        P2PTest* p2pTest = P2PTest::getInstance();

        // Atualiza tela para que o usuário saiba que algum amigo mudou de status.
        ...
}
```

4 Responsáveis pelo Documento

Nome: Cesar Augusto Camillo Teixeira

E-mail: cesar@dc.ufscar.br

Telefone: (0xx16) 3351-8614

Instituição: Laboratório para Inovação em Computação e Engenharia da Universidade Federal de São Carlos (Lince-UFSCar)

Nome: Maria da Graça Campos Pimentel

E-mail: mgp@icmc.usp.br

Telefone: (0xx16) 3373-9657

Instituição: Laboratório Intermidia do Instituto de Computação e Matemática computacional da Universidade de São Carlos (Intermidia - ICMC/USP)

Nome: Erick Lazaro Melo

E-mail: erick_melo@dc.ufscar.br

Telefone: (0xx16) 3351-8614

Instituição: Laboratório para Inovação em Computação e Engenharia da Universidade Federal de São Carlos (Lince-UFSCar)