UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ISRAEL ANDREIS

Estudo Comparativo entre Bibliotecas de Implementação para o Protocolo SIP

Trabalho de Graduação

Prof. Dr. Valter Roesler
Orientador

Porto Alegre, dezembro de 2010

"Não cruze os braços diante de uma dificuldade, pois o maior homem do mundo morreu de braços abertos!"

— BOB MARLEY

AGRADECIMENTOS

Gostaria de agradecer ao Prof. Valter Roesler por me orientar durante este trabalho. Agradeço também a todos os professores que me ajudaram durante minha graduação, eles são os responsáveis pelo conhecimento adquirido. Obrigado ao pessoal do laboratório de Projetos em Áudio e Vídeo pelas sujestões e incestivos prestados neste e em outros trabalhos: ao Felipe pelas dicas e conselhos, ao Leonardo Borba pela amizade e motivação, ao Leonardo Daronco e ao Daniel pelas dicas e experiência e por fim ao Paulo, Mateus e a Luiza pela ajuda.

Quero prestar um agradecimento especial à minha família que me apoio durante a graduação e a todas outras pessoas que direta ou indiretamente me ajudaram durante estes anos. Muito Obrigada!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CPL	Common Public License
FAQ	Frequently Asked Questions
GPL	General Public License
GQM	Goal-Question-Metric
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IM	Instant Messaging
IP	Internet Protocol
ISO/IEC	International Organization for Standardization/Internacional Eletronical Commission
ITU-T	International Telecommunication Union
LGPL	Lesser General Public License
MIME	Multipurpose Internet Mail Extensions
MPL	Mozilla Public License
NAT	Network Address Translation
OS	Operating System
OSI	Open Systems Interconnection
QoS	Quality of Service
RFC	Request For Comment
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SAP	Session Announcement Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Indicators
VoIP	Voice over Internet Protocol

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A utilização de softwares de videoconferência e VoIP vem se difundindo atualmente. Para que softwares deste tipo, desenvolvidos por empresas diferentes, possam ter interroperabilidade é necessário que sigam um padrão de comunicação. Para este fim existem os protocolos de sinalização. O protocolo de sinalização SIP vem sendo bastante utilizado devido a sua simplicidade em relação aos demais. Inúmeras bibliotecas de funções para implementação do protocolo SIP vem sendo desenvolvidas para a difusão deste padrão. Logo, este trabalho realiza um estudo comparativo entre algumas destas bibliotecas considerando características qualitativas referentes aos seus projetos e características quantitativas focadas na medida de esforço necessário para criação de um aplicativo multimídia baseado no protocolo SIP.

Palavras-chave: VoIP, videoconferência, protocolo de sinalização, SIP, comparação, bibliotecas, multimídia.

Comparative Study of Libraries for Implementing the SIP Protocol

ABSTRACT

The use of videoconferencing and VoIP software is spreading today. For such software, developed by different companies may have interoperability is necessary to follow a pattern of communication. For this purpose there are signaling protocols. The SIP signaling protocol has been widely adopted due to its simplicity compared to the others. Many libraries of functions to implement the SIP protocol has been developed to spreading this standard.Therefore, this paper conducts a comparative study between some of these libraries considering qualitative characteristics related to their projects and quantitative traits focused on the extent of effort required to create a multimedia application based on SIP protocol.

Keywords: VoIP, videoconferencing, signaling protocols, SIP, comparison, libraries, multimedia.

1 INTRODUÇÃO

Nos últimos anos, a maneira com que a comunicação entre as pessoas é feita, vem sofrendo mudanças consideráveis. As comunicações "face-a-face"têm sido substituídas por outros meios de comunicação como telefone e principalmente o computador. As novas tecnologias de computação e de comunicação causam mudanças na quantidade, na qualidade e na velocidade com que as informações são trocadas.

A utilização de sistemas de videoconferência, de VoIP (Voice over Internet Protocol) e de IM[1] vem se difundindo largamente na sociedade atual. A popularização de um grande número de softwares que prevêem estes serviços reforça esta afirmação. Como exemplos destes softwares podem ser citados: X-Lite, Skype, Windows Messenger, Microsoft NetMeeting, Polycom PVX, entre outros.

Uma série de empresas desenvolve softwares para comunicação entre usuário. No entanto, para que estes softwares, desenvolvidos por grupos diferentes, consigam ter interoperabilidade é necessário que sigam um padrão de comunicação. Para este fim, existem protocolos de sinalização.

Um protocolo de sinalização permite que sistemas multimídia diferentes possam se comunicar através de uma linguagem comum. Existem vários protocolos de sinalização, mas em geral, sistemas de VoIP e de videoconferência dão suporte a apenas um ou dois destes (WALLINGFORD, 2005). Atualmente, os dois principais protocolos de sinalização utilizados são o SIP (Session Initiation Protocol), que é um padrão da IETF, e o padrão H.323 que é parte da família de recomendações da ITU-T.

O padrão H.323, desenvolvido inicialmente em junho 1996, especifica o uso de áudio, vídeo e dados em comunicações multimídia utilizando redes baseadas em pacotes. Além disso, estabelece padrões para a codificação e decodificação de fluxos de dados de áudio e vídeo, tendo como objetivo garantir a interoperabilidade de produtos baseados em sua especificação (LEOPOLDINO; MEDEIROS, 2001). O padrão H.323 é largamente usado desde sua criação, mas um dos fatores negativos de sua utilização é sua grande complexidade e a dificuldade de extensão.

O protocolo SIP foi desenvolvido inicialmente como uma forma de sinalização multiusuário distribuída para aplicações de telefonia e troca de mensagens em uma rede IP (WALLINGFORD, 2005). Desenvolvido com o intuito de ser simples e extensível, nos últimos anos, o SIP vem se tornando uma alternativa ao uso do H.323 para a construção de soluções de videoconferência que utilizam rede IP.

A utilização um protocolo de sinalização com certeza não é uma tarefa fácil. Programar um software que siga a risca as regras dos padrões já estabelecidos exige um trabalho árduo e propenso a erros. Contudo, existem bibliotecas de funções que implementam es-

[1] IM: Instant Messaging, nomenclatura em inglês para troca de mensagens instantâneas.

tes protocolos. O objetivo disto é facilitar a tarefa de um programador que deseja escrever sua aplicação utilizando um padrão de sinalização. A Escolha adequada de uma biblioteca que preencha os requisitos necessários para se escrever uma aplicação nem sempre é fácil.

Tendo em vista que o protocolo SIP vem sendo utilizado em softwares de videoconferência, este trabalho de conclusão pretende comparar algumas das principais bibliotecas que implementam este protocolo. O resultado deste estudo comparativo servirá como uma boa referência para a comunidade de programadores que deseja utilizar uma biblioteca que implemente o protocolo SIP.

1.1 Motivação

A motivação para este projeto vem basicamente de dois fatores: o primeiro é satisfação pessoal, pois venho trabalhando em um sistema de videoconferência a mais de dois anos e a integração de um protocolo de sinalização ao sistema contribuiria em muito para o seu desenvolvimento. O segundo é que em minha visão, a videoconferência, em um futuro próximo, se tornará o principal modo de comunicação à distância, sendo este um bom motivo para ampliar meus conhecimentos nesta área.

Durante o projeto desenvolvido junto ao laboratório de Projetos em Áudio e Vídeo (PRAV) na Universidade Federal do Rio Grande do Sul, tive a oportunidade de estudar e aprender muito sobre videoconferência e sistemas para ensino a distância. Durante este período, meu conhecimento sobre protocolos de sinalização foi enriquecido, em especial sobre o protocolo SIP, por isso este tema foi escolhido com proposta de trabalho de conclusão.

A recente vontade de voltar minha formação para a área de redes também motiva a conclusão deste trabalho, que estará abrangendo uma área bastante significante de comunicação de dados na rede IP.

O desafio e o aprendizado são dois fatores muito importantes. A conclusão deste projeto provê uma satisfação muito grande como cientista e como pessoa. Ao mesmo tempo este trabalho poderá prover um diferencial como profissional.

1.2 Objetivo

O objetivo deste trabalho de conclusão de curso consiste em um estudo comparativo entre algumas das principais bibliotecas de implementação para o protocolo SIP. Para este estudo serão levados em conta alguns critérios para avaliação de qualidade de software presentes na literatura na área de engenharia de software.

O objetivo específico deste estudo se dá através da adaptação de algumas funcionalidades do protocolo SIP a um software de videoconferência já existente. Este software dispõe de boa parte dos requisitos necessário para realizar uma videoconferência, excetuando-se um protocolo de sinalização. Através desta implementação se espera validar os conceitos teóricos e práticos desenvolvidos durante o trabalho.

O critério de escolha das bibliotecas é baseado nos requisitos que o software base impõe. A partir disso, serão utilizadas métricas qualitativas e quantitativas no intuito de obter resultados subjetivos e objetivos das bibliotecas comparadas. Ao final, serão gerados gráficos contendo os resultados do estudo que se baseia na avaliação dos seguintes quesitos:

- Usabilidade.

- Documentação.
- Suporte.
- Métricas de código

A tarefa de escolher uma biblioteca de funções que implemente o protocolo SIP, dentre as inúmeras disponíveis, não é simples nem mesmo para quem já tem uma boa experiência com SIP. Uma contribuição científica deste trabalho visa analisar algumas destas bibliotecas de forma imparcial e disponibilizar para a comunidade um estudo objetivo sobre os pontos fracos e pontos fortes de cada uma das bibliotecas analisadas.

1.3 Estrutura

A estrutura do trabalho segue da seguinte forma.

No capítulo 2, **SIP**, é introduzida a pilha do protocolo SIP. Neste capítulo são descritos os protocolos envolvidos no padrão SIP e como eles interagem entre si para que uma aplicação possa utilizá-la de maneira a satisfazer suas necessidades. A utilização destes protocolos, bem como suas funcionalidades, é abordada de maneira simples para que os leitores deste trabalho possam entender como este padrão opera.

No capítulo 3, **Bibliotecas SIP**, são citadas as principais bibliotecas de funções que implementam o protocolo SIP. Este capítulo tem como objetivo descrever brevemente as bibliotecas disponíveis atualmente citando suas principais características. As características desejáveis em uma biblioteca SIP e os fatores que determinaram quais bibliotecas serão utilizadas neste trabalho também estão contidos neste capítulo.

O capítulo 4, **Metodologia**, descreve a metodologia que será seguida durante o desenvolvimento do trabalho. As métricas escolhidas e suas aplicações na comparação das bibliotecas escolhidas são descritas neste capítulo.

O capítulo 5, **Implementação**, descreve como as bibliotecas SIP foram utilizadas na adaptação ao aplicativo. Informações relevantes a implementação prática e possíveis dificuldades encontradas durante esta etapa são mencionadas neste capítulo.

O capítulo 6, **Validação e Resultados**, descreve os resultados obtidos durante o trabalho. A validação da metodologia utilizada para comparação das bibliotecas é desenvolvida neste capítulo, e assim, fornece uma base teórica e prática que relaciona as vantagens e desvantagens de cada biblioteca estudada.

Por fim, o capítulo 7 trás as Conclusões deste trabalho, descrevendo os principais pontos e as considerações finais sobre o estudo realizado ao longo deste. Em seguida, são apresentadas as referências bibliográficas utilizadas e uma última seção com apêndices sobre assuntos secundários.

2 SIP

Existem muitos aplicativos da Internet que exigem a criação e a gerência de uma sessão, onde uma sessão é considerada como uma troca de dados entre um conjunto de participantes. A implementação destas aplicações é complexa pelas possíveis práticas dos participantes: os usuários podem utilizar terminais[1] diferentes para ter acesso à rede, podem ser endereçados por vários nomes, e eles podem se comunicar através de mídias diferentes - às vezes simultaneamente (ROSENBERG et al., 2002).

Uma variedade de protocolos foi criada para transportar e manipular as várias formas de dados multimídia de uma sessão, como voz, vídeo ou mensagens de texto. O Session Initiation Protocol (SIP) trabalha em conjunto com estes protocolos, permitindo aos terminais (chamados de *User Agents*) descobrirem uns aos outros e chegarem a um acordo sobre qual caracterização de sessão gostariam de compartilhar. Para localizar potenciais participantes de uma sessão, e para outras funções, o SIP permite a criação de uma infraestrutura de servidores de rede (chamados de *Proxy Servers*) para que os *User Agents* possam enviar registros, convites para sessões, entre outras solicitações. O SIP é uma ferramenta ágil e de uso geral para criar, modificar e terminar sessões que funcionam independentemente dos protocolos de transporte e do tipo de sessão que está sendo estabelecida (ROSENBERG et al., 2002).

2.1 Descrição do Protocolo

O SIP é um protocolo utilizado para estabelecer chamadas e conferências através de redes via IP, e atua na camada 7 do modelo OSI (TANENBAUM, 2003), a camada de aplicação. Além disso, ele foi projetado tendo como foco a simplicidade, e, como um mecanismo de estabelecimento de sessão, ele apenas inicia, termina e modifica a sessão, o que o torna um protocolo que se adapta facilmente a diferentes arquiteturas (ROSENBERG et al., 2002). O SIP oferece cinco tipos de serviços para iniciação e finalização de sessões multimídias, descritas abaixo:

- Localização do usuário: responsável pela localização do terminal para estabelecer a conexão
- Disponibilidade do usuário: responsável por realizar a vontade do usuário em estabelecer uma sessão de comunicação.
- Recursos do usuário: responsável pela determinação dos meios de comunicação e os parâmetros a serem utilizados.

[1] terminal: também conhecido como endpoint, é nome dado a aplicação que um usuário utiliza para participar de uma sessão.

- Gerência da sessão: responsável por inicializar, terminar ou colocar em espera uma sessão.
- Modificar sessão: responsável por modificar uma sessão em andamento.

Para estabilizar uma sessão multimídia o SIP faz uso do Session Description Protocol (SDP) que é um protocolo para a descrição dos parâmetros de inicialização de mídia *streaming*. O SDP não entrega a mídia em si, mas é usado para a negociação entre os pontos finais de tipo de mídia, o formato, e todas as propriedades associadas. O conjunto de propriedades e parâmetros são muitas vezes chamado de perfil da sessão, além disso, o SDP foi projetado para ser extensível e suportar novos tipos de mídia e formatos.

O transporte de mídia é feito através do uso do Real-time Transport Protocol (RTP). Este protocolo define um formato de pacote padronizado para a entrega de áudio e vídeo através da Internet e é amplamente utilizado em sistemas de comunicação e entretenimento que envolve mídia *streaming*, tais como telefonia e videoconferência. Em geral, o RTP é utilizado em conjunto com o RTP Control Protocol (RTCP), enquanto o RTP carrega os fluxos de mídia, por exemplo, áudio e vídeo, o RTCP é usado para monitorar as estatísticas de transmissão e informações de qualidade de serviço (QoS).

A figura 2.1 ilustra a pilha de protocolos utilizada pelo padrão SIP para que seu uso se torne possível.

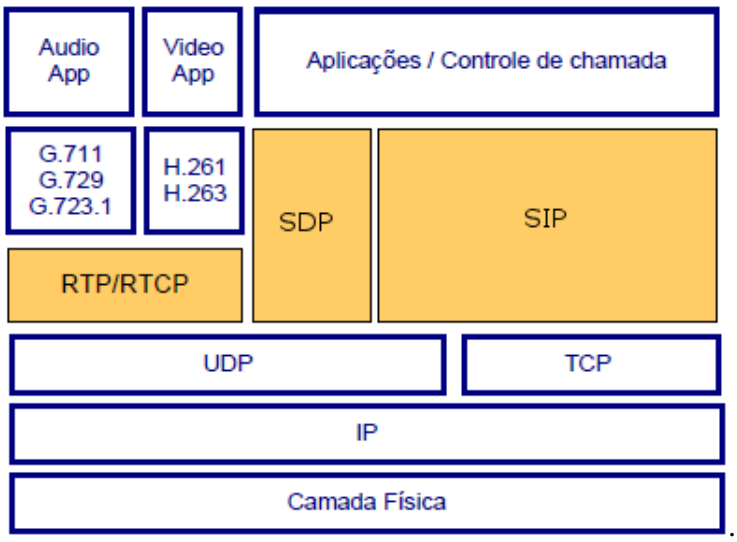

Figura 2.1: Pilha de SIP e protocolos envolvidos

2.2 SDP: Session Description Protocol

Ao iniciar videoconferências, chamadas VoIP, *streaming* de vídeo ou outras sessões, é necessário apresentar os detalhes de mídia, endereços de transporte, e outros metadados de uma sessão para seus participantes. Especificado na *Request For Comment* 4566 (HANDLEY et al., 2006), o *Session Description Protocol* (SDP) fornece um padrão para a representação dessas informações, independentemente da forma como essa informação é transportada. O SDP é puramente um formato de descrição de sessão, não incorpora um protocolo transporte, e destina-se a utilizar protocolos de transporte diferentes, incluindo Session Announcement Protocol (SAP), Session Initiation Protocol (SIP), Real

Time Streaming Protocol (RTSP), correio eletrônico usando as extensões MIME e Hypertext Transport Protocol (HTTP) (HANDLEY et al., 2006).

Descrições de sessões utilizadas no protocolo SIP fazem uso do protocolo SDP. As mensagens SIP, usadas para criar sessões, carregam uma descrição de sessão que permite aos participantes concordar com um conjunto de tipos de mídia compatível. Quando usado com SIP, o modelo de oferta/resposta fornece um quadro limitado de negociação utilizando SDP (HANDLEY et al., 2006).

O objetivo do SDP é transmitir informações sobre os fluxos de mídia que cada terminal disponibiliza para estabelecer uma sessão multimídia. Um terminal SIP envia uma descrição SDP quando deseja convidar um outro terminal para uma sessão. Como resposta recebe uma outra descrição contendo os tipos de mídia acordados pelo outro terminal, assim, a negociação é concluída e ambos terminais já sabem que mídia será utilizada.

A figura 2.2 exemplifica o corpo de uma mensagem SDP:

```
v=0
o=Israel_Note 3496292970 3496294380 IN IP4 192.168.0.132
s=tcc-demonstration
c=IN IP4 192.168.0.132
t=0 0
m=audio 14000 RTP/AVP 96
a=rtcp:14001 IN IP4192.168.0.132
a=rtpmap:96 MPA/90000
a=fmtp:96 layer=2;bitrate=256000;samplerate=48000;mode=stereo
a=sendrecv
m=video 14002 RTP/AVP 98
a=rtcp:14003 IN IP4 192.168.0.132
a=rtpmap:98 MP4V-ES/90000
a=fmtp:98
a=sendrecv
```

Figura 2.2: Exemplo de mensagem SDP

Neste exemplo, o campo "v=0"descreve a versão do protocolo. A segundo linha, que contém o identificador "o=", indica a origem da mensagem e o identificador de sessão. O campo "s=tcc-demostration"representa o nome da sessão. A linha "c=IN IP4 192.168.0.132"apresenta informações sobre a conexão. O campo "t=0 0"indica o tempo de início e fim de uma sessão, neste caso, quando o tempo de fim de sessão é igual a zero, representa que a sessão não tem tempo determinado para sua finalização. Os campos que iniciam com o identificador "m="descrevem o tipo de mídia utilizado, a porta em que se espera receber o fluxo de mídia, o protocolo de transporte utilizado pelo tipo de mídia e por fim o codec[1] em que a mídia utiliza. Os campos identificados com "a="são atributos que descrevem a mídia identificada anteriormente pelo identificador "m=", estes atributos visam descrever informações adicionais sobre o fluxo de mídia descrito.

2.3 RTP: Real-time Transport Protocol

A utilização do protocolo UDP é adequada para transporte de tráfego multimídia onde o reenvio de dados não se faz necessário, já que possíveis perdas de dados são suportadas

[1]codec: é o acrônimo de Codificador/Decodificador, dispositivo de hardware ou software que codifica/-decodifica sinais.

por sistemas fim-a-fim. Em contra partida, o UDP não fornece ao usuário uma configuração de parâmetros de requisito de largura de banda, prejudicando o serviço quando ocorre congestionamento na rede. Para integrar este protocolo com tráfego multimídia, alguns protocolos que atuam em camadas superiores foram propostos, dentre eles o Real-Time Transport Protocol (RTP) e o RTP Control Protocol (RTCP).

Especificado na *Request For Comment* 3550 (RFC 3550), o *Real-Time Transport Protocol* (RTP) fornece a especificação de serviços de entrega fim-a-fim para dados com características de tempo real, como áudio e vídeo iterativo. Esses serviços incluem identificação de tipos de dados, numeração seqüencial de dados, temporização de dados e acompanhamento de entrega sobre o protocolo UDP (SCHULZRINNE et al., 2003).

O RTP por si só não providencia nenhum mecanismo que garanta entrega de dados ou qualidade de serviço (QoS), pois depende de serviços disponibilizados pelas camadas inferior da rede. O RTP faz uso do RTP Control Protocol (RTCP) descrito em (SCHULZRINNE et al., 2003) para monitoramento de QoS e para transmitir informações sobre os participantes em uma sessão em curso.

2.4 Funcionamento do Protocolo

2.4.1 Entidades SIP

O protocolo SIP define diversas entidades que juntas compõe sua arquitetura. Os quatro principais componentes dessa arquitetura são os *User Agents*, *Proxy Server*, *Redirect Server* e *Registrar Server*. O papel de cada uma destas entidades é importante para que um sistema possa utilizar todas as funcionalidades da arquitetura SIP (CAMARILLO, 2001). A descrição abaixo resume o papel que cada entidade desempenha dentro do protocolo:

- *User Agents*: está é a entidade SIP que interage com o usuário. Possui a capacidade de enviar e receber requisições, assim, ele pode agir tanto como cliente (*User Agents Client*), enviando requisições e recebendo respostas, ou como servidor (*User Agents Server*), enviando respostas e recebendo requisições.

- *Proxy Server*: É uma entidade intermediária que atua tanto como servidor e como cliente com a finalidade de fazer pedidos em nome de outros clientes. Um *Proxy Server* tem o papel de roteamento, o que significa que sua tarefa é assegurar que a solicitação é enviada para outra entidade para que possa chegar ao usuário-alvo. Além disso, se necessário, o *Proxy Server* pode reescreve partes específicas de uma mensagem de pedido antes de encaminhá-lo.

- *Redirect Server*: É um tipo de servidor SIP que ajuda a localizar SIP *User Agents*, fornecendo localizações alternativas, onde o usuário pode ser acessado.

- *Registrar Server*: É um servidor que armazena registros sobre usuários, fornecendo um serviço de localização.

2.4.2 Métodos SIP

O SIP funciona numa arquitetura cliente/servidor e em sua versão atual especificada em (ROSENBERG et al., 2002) define um total de seis métodos de requisição, *INVITE*, *ACK*, *CANCEL*, *OPTIONS*, *REGISTER* e *BYE*, que serão descritos a seguir.

2.4.2.1 INVITE

Este método é usado para iniciar sessões e divulgar a capacidade dos terminais. O corpo das mensagens de pedidos de *INVITE* contém a descrição da sessão. Por exemplo, quando o usuário UserA convida o usuário UserB, seu *User Agent* envia uma mensagem de *INVITE* com a descrição da sessão para o *User Agent* de UserB. Supondo a utilização de SDP para descrever a sessão, o usuário UserB receberá a seguinte descrição de sessão ilustrada na figura 2.3:

```
v=0
o=João 2890844526 2890842807 IN IP4 143.54.132.112
s=Demonstração
c=IN IP4 143.54.132.112
t=0 0
m=audio 49170 RTP/AVP 1
a=rtpmap:1 G726/8000
m=video 51372 RTP/AVP 31
a=rtpmap:31 H261/90000
```

Figura 2.3: Uso de SDP no método INVITE

O *INVITE* recebido pelo usuário Pedro significa que João está convidando Pedro para participar de uma sessão de áudio e vídeo. A partir da descrição da sessão realizada no *INVITE*, o *User Agent* de Pedro sabe que João deseja receber pacotes RTP contendo áudio no endereço 143.54.132.112 na porta de número 49170 e pacotes RTP contendo vídeo no mesmo endereço, mas na porta número 51372. Além disso, o *User Agent* de Pedro sabe também que para realização da sessão o *User Agent* de João suporta G.726 como codec de áudio e H.261 como codec de vídeo.

O *User Agent* de Pedro alerta o mesmo, indicando a existência de uma chamada. No mesmo instante, envia uma resposta *"180 Ringing"* para o *User Agent* de João indicando que a chamada de inicialização de sessão está em curso e aguarda uma confirmação de Pedro para que a sessão continue. Quando Pedro finalmente aceitar a chamada, uma resposta *"200 OK"* é enviada para a *User Agente* de João indicando que a chamada foi aceita e junto com a resposta é enviada a descrição de sessão de Pedro, indicando quais codec de áudio e vídeo serão usados durante a sessão e informando seu endereço e portas para o recebimento de mídia.

A figura 2.4 ilustra a troca de mensagem *INVITE* dos participantes da sessão:

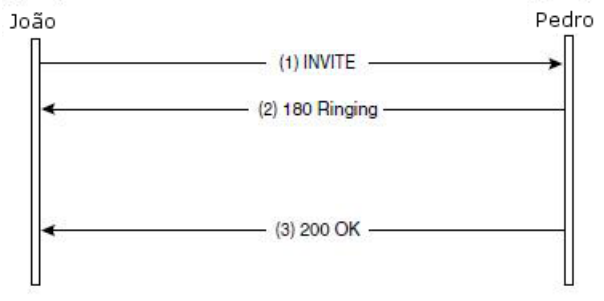

Figura 2.4: Exemplo de uso do método INVITE

2.4.2.2 *ACK*

Este método é utilizado para notificar a confirmação de um *INVITE*. Assim, o cliente de origem de um pedido de *INVITE* emite uma confirmação *ACK* quando recebe uma resposta final de outro participante. A figura 2.5 ilustra a troca de mensagem ACK entre os participantes.

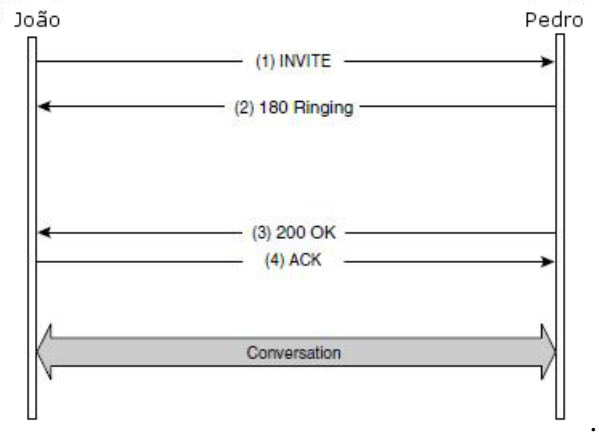

Figura 2.5: Exemplo de uso do método ACK

2.4.2.3 *CANCEL*

Este método é utilizado durante as tentativas de substituir um pedido prévio que ainda não foi concluído. Por exemplo, se o João está tentando estabelecer uma sessão com o Pedro, mas com o passar do tempo João não recebeu uma resposta final de Pedro e decide finalizar a tentativa de estabilização de sessão, uma mensagem *CANCEL* deve ser enviada. A figura 2.6 ilustra a troca de mensagem *CANCEL* entre os participantes.

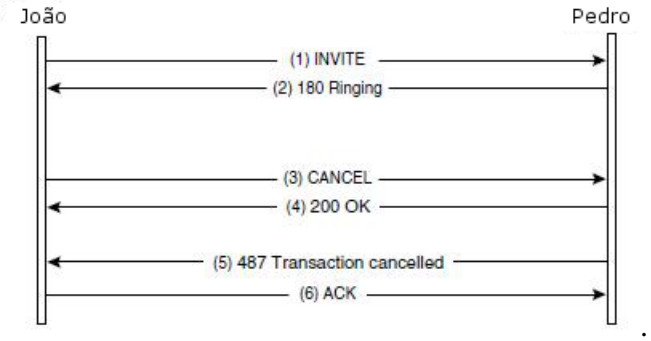

Figura 2.6: Exemplo de uso do método CANCEL

2.4.2.4 *OPTIONS*

Este método é usado para consultar a informação de capacidade de um terminal, sem realmente estabelecer um canal de mídia. As capacidades incluem quais métodos SIP e quais protocolos de descrição de sessão são suportados. A figura 2.7 ilustra a troca de mensagem *OPTIONS* entre dois participantes.

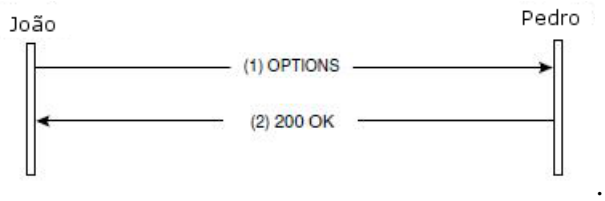

Figura 2.7: Exemplo de uso do método OPTIONS

2.4.2.5 REGISTER

Usuários enviam pedidos *REGISTER* para informar um servidor (neste caso, referido como um registrador) sobre a sua localização atual. Por exemplo, João pode enviar um *REGISTER* para um *Registrar Server* no domínio específico informando que todas as solicitações que João receber devem ser redirecionadas seu novo endereço. A figura 2.8 ilustra um cenário em que uma mensagem *REGISTER* é enviada.

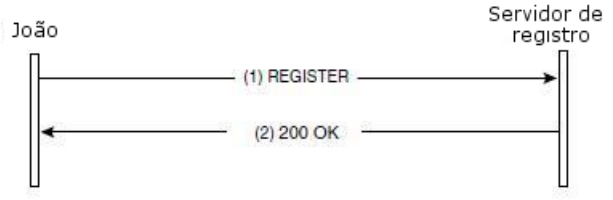

Figura 2.8: Exemplo de uso do método REGISTER

2.4.2.6 BYE

Este método ocorre quando uma sessão for completada, ou seja, pelo menos um usuário deseja deixar a sessão. Por exemplo, Pedro deseja finalizar uma sessão com em andamento com João, para isso ele envia uma mensagem BYE indicando que a sessão deve ser terminada. A figura 2.9 ilustra um cenário em que uma mensagem *BYE* é enviada.

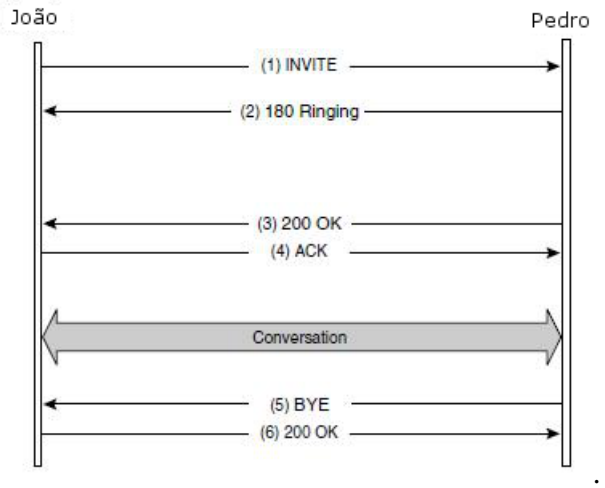

Figura 2.9: Exemplo de uso do método BYE

2.4.3 Respostas SIP

Quando uma chamada for iniciada, terminada, ou alterada, um método SIP é invocado. Os métodos SIP na listados anteriormente são semelhantes em conceito aos métodos *GET* e *POST* do protocolo HTTP, e como o HTTP, o SIP espera códigos de resposta quando envia um método (WALLINGFORD, 2005). Códigos numéricos SIP de resposta são três dígitos e se dividem em seis categorias que estão listadas abaixo:

- 1xx - Informational: contém mensagens que indicam o progresso da chamada, a primeira mensagem dessa classe recebida pelo *User Agent*, confirma o recebimento do *INVITE* pelo servidor e indica que o *User Agent* deve parar de enviar solicitações *INVITE*.
- 2xx - Success: indica a aceitação de uma solicitação.
- 3xx - Redirection: indica que o destinatário não está nessa localização e em geral informa sua nova a localização. Em geral esta resposta é enviada por um *Redirect Server*.
- 4xx - Failure: informa que houve uma falha na solicitação por parte de um cliente, e que o mesmo deverá refazer a solicitação de acordo com o falha indicada.
- 5xx - Error: informa que a mensagem não pode ser enviada com sucesso em decorrência de um erro com um servidor, o cliente poderá fazer essa mesma solicitação em outros servidores.
- 6xx - Availability: informa que houve um erro e que essa mensagem irá falhar em qualquer servidor, portanto não deverá ser reenviada.

A lista completa com todas as respostas SIP pode ser encontrada na tabela do ANEXO A.

3 BIBLIOTECAS SIP

Visto que a criação de uma biblioteca que implemente funcionalidades do protocolo SIP é trabalhosa e sujeita a erros, a utilização de uma solução já difundida pode ser a escolha mais adequada. Assim, este capítulo tem como objetivo descrever algumas das principais bibliotecas SIP (também conhecidas com pilhas SIP) difundidas através da internet e selecionar algumas, que satisfaçam alguns critérios especificados, para um estudo comparativo.

3.1 Características Desejadas

Qualquer desenvolvedor que depare com a tarefa de utilizar uma biblioteca que implemente o protocolo SIP terá grande dificuldade em tentar selecionar algumas delas, mesmo que seja um especialista em SIP, pois cada pilha apresenta pontos fracos e fortes (FARIAS et al., 2008). Tomando com base a documentação disponibilizada pelos seus desenvolvedores é possível ter uma idéia de como cada biblioteca funciona, mesmo assim, não é possível determinar sua qualidade apenas levando em conta este fator.

Uma biblioteca que implementa uma pilha de protocolos SIP deve, inicialmente, garantir a interoperabilidade entre os demais elementos da arquitetura SIP existente. A interoperabilidade garante que aplicações que utilizem esta pilha tenham a capacidade de criar, estabilizar, alterar e finalizar uma sessão de acordo com a especificação (ROSENBERG et al., 2002). Além disso, é necessário que a biblioteca ofereça uma interface de programação para que aplicações construídas a partir dela possam ser criadas.

Um atributo importante e desejável em uma biblioteca se diz respeito a estabilidade. Tendo em vista que o trabalho de construção de uma pilha SIP é enorme e sujeito a erros, a utilização de uma pilha que possua um projeto estável e com um bom número de adeptos se torna um requisito interessante do ponto de vista da continuidade da pilha e da facilidade de detecção (e posterior correção) de erros que um maior número de usuários possibilita (FARIAS et al., 2008).

A validação do estudo comparativo entre bibliotecas de implementação do protocolo SIP será feita através da adaptação de algumas funcionalidades do protocolo SIP a um software já existem de videoconferência. Isso servirá para que o estudo teórico desenvolvido durante o trabalho seja aliado a prática e construa resultados mais precisos. Além de estar de acordo com a especificação (ROSENBERG et al., 2002), é desejável que a pilha analisada seja capaz de criar uma sessão, efetuar a negociação de mídia, estabilizar uma sessão e por fim encerrar uma sessão. Assim, a adaptação prática utilizará diferentes bibliotecas de funções que se encaixem nos requisitos descritos.

O software que será utilizado como base para adaptação do protocolo SIP é implementado na plataforma Windows XP, portanto é necessário que a biblioteca utilizada ne-

cessariamente de suporte a este sistema operacional. Este característica descarta algumas pilhas SIP disponíveis que foram desenvolvidas com o objetivo de dar suporte a apenas um ou dois sistemas operacionais, mesmo assim a maioria delas suporta grande parte das plataformas existentes, assim sendo, são utilizadas por uma gama maior de desenvolvedores.

Outro critério essencial é a linguagem de programação utilizada pelas bibliotecas. O software utilizado é escrito na linguagem C++, mesmo assim, suporta módulos criados a partir da linguagem C, então, serão analisadas somente as bibliotecas que tenham sido desenvolvidas a partir da linguagem de programação C ou C++.

Por fim, devido ao fato de que para um estudo comparativo entre diferentes bibliotecas muitas vezes é necessário a análise de código fonte, serão analisadas somente bibliotecas de implementação *Open Source*.

Como descrito anteriormente, foram determinados critérios de escolha para selecionar algumas bibliotecas SIP que possam ser analisadas durante este trabalho. Os critérios definidos servirão para que o estudo comparativo realizado durante este trabalho possa ser validado a partir de uma adaptação do protocolo SIP a uma aplicação de videoconferência já existente. Abaixo serão listados os critérios utilizados para ilustra melhor as escolhas:

- A biblioteca deve estar de acordo com especificação (ROSENBERG et al., 2002);
- Deve possibilitar ao programador funções para criar uma sessão, efetuar a negociação de mídia (disponibilizar negociação SDP), estabilizar uma sessão e por fim encerrar uma sessão;
- Ser Open Source;
- Ter suporte a plataforma Windows XP ou superior;
- Ser escrita nas linguagens C ou C++;
- Ter estabilidade, ou seja, estar em desenvolvimento e apresentar release contínuos para que possíveis erros e bugs possam ser corrigidos.

3.2 Bibliotecas Disponíveis

Para a difusão e evolução do protocolo SIP, vários grupos têm trabalhado no desenvolvimento de bibliotecas SIP abertas, que podem ser usadas para o desenvolvimento de aplicações mais avançadas que incorporem as facilidades e novos métodos que vêm sendo incorporados ao SIP (FARIAS et al., 2008). Efetuando uma pesquisa no Google[1] serão descritas inúmeras bibliotecas SIP *Open Source* existentes, dentre elas destacam-se: eXoSIP, JAIN-SIP, minisip, MjSIP, OpenSIPStack, oSIP library, PJSIP, ReSIProcate e Sofia-SIP.

As bibliotecas citadas apresentam diferentes características em relação aos sistemas operacionais em que operam, linguagem de programação em que foram escritas, tipos de licença, entre outras. A tabela 3.1 exemplifica estas características.

De acordo com a tabela 3.1, nem todas as bibliotecas citadas atende os requisitos especificados na sessão 3.1. As subseções a seguir descrevem algumas características que dêem suporte ao sistema operacional Windows e sejam escritas nas linguagens C ou C++.

[1]Google: http://www.google.com/

Tabela 3.1: *Características das pilhas SIP*

Nome	Plataformas Suportadas	Linguagem	Desenvolvedor	Licença
eXoSIP2	GNU/Linux, MacOs, Windows e Windows Mobile	C	Antisip, (Aymeric Moizard)	GPL
NIST/JAIN-SIP	Plataforma Java e J2ME	Java	NIST(National Institute of Standards and Technology) , apoio da Sun MycroSystems	CPL
Minisip	Linux, Windows (XP/2000), Pocket PC	C++	Desenvolvido por Doutores e Mestres da Royal Institute of Technology(KTH, Estocolmo, Suécia) e voluntários	GPL
MjSIP	Plataforma Java e J2ME	Java	mjsip.org	GPL
OpenSIPStack	GNU/Linux, Solaris, BSD, Darwin, Windows	C++	OpenSIPStack.org	MPL
oSIP Library	GNU/Linux, OpenDsd 3.1/3.2, Windows(NT/ 95/2000), Solaris, HP-Unix	C	GNU Project, (Aymeric Moizard)	LGPL
PJSIP	GNU/Linux, MacOs, Windows, Windows Mobile, RTEMS, Symbian OS	C	pjsip.org	GPL
ReSIProcate	FreeBSD, Linux, QNX, SunOS, MacOs, Windows	C++	resiprocate.org	Vovida License
Sofia-SIP	GNU/Linux, MacOs, Windows	C	Nokia Research Center	LGPL

3.2.1 oSIP Library

oSIP library é uma biblioteca escrita em C que visa proporcionar, aos desenvolvedores de softwares de comunicação multimídia, uma interface simples e poderosa baseadas no protocolo SIP (MOIZARD, 2008a). Desenvolvida por Aymeric Moizard, começou a ser desenvolvida em setembro de 2000 e apresenta suporte a várias plataformas, dentre elas: GNU/Linux, MacOSX (Darwin), OpenBsd 3.1/3.2, Windows NT/95/2000, Solaris, HP-Unix.

Depois de uma série de ajustes visando tornar sua solução de mais fácil utilização aos usuários, a oSIP passou se chamar oSIP2. Apesar disso, na documentação disponibilizada ao se obter os arquivos do projeto, o autor avisa que a biblioteca ainda apresenta problemas de complexidade em sua utilização.

A versão atual da biblioteca é 3.3.0, disponibilizada em 04 de março de 2009 e apresenta licença LGPL[1]. Contudo, ainda em 2003,seu desenvolvedor passou a criar uma nova biblioteca chamada eXosip, que é a extensão do projeto oSIP.

[1]Disponível em <http://www.gnu.org/licenses/lgpl.html>, acesso em novembro de 2010.

3.2.2 eXoSIP2

A biblioteca eXosip2 é uma extensão da biblioteca oSIP e tem como objetivo implementar uma API simples para controlar a utilização do protocolo SIP. Desenvolvida na linguagem de programação C, apresenta licença de uso GLP[1] e é mantida por Aymeric Moizard (e sua empresa, Antisip), mesmo criador da biblioteca oSIP. Em sua última atualização, em março de 2009, apresenta suporte para as plataformas Linux, MacOs, Windows e Windows Mobile.

Apesar de implementar grande parte da especificação do protocolo SIP, a eXosip2 não implementa funções que dêem suporte ao protocolo de transporte RTP e negociação SDP (MOIZARD, 2008b).

3.2.3 OpenSipStack

A biblioteca OpenSIPStack é escrita em C++ e é mantida pelo projeto OpenSIPStack.org fundado e coordenado por Joegen E. Baclor. O trabalho teve início em janeiro de 2005 e está disponível sobre as licenças MPL[2], GPL e LGPL, tendo como licença ativa padrão MPL, e as licenças GPL e LGPL como alternativas.

OpenSIPStack funciona em diversas plataformas (GNU/Linux, Solaris, BSD, Darwin e Windows), e disponibiliza suporte tanto para o protocolo SIP quanto para tratamento de mídias de áudio. Segundo (BACLOR, 2005), o suporte a mídias de vídeo está planejado para versões futuras. Um ponto forte da biblioteca é a disponibilidade de mecanismos de tratamento para transmissão de mídia e o suporte a vários codecs de áudio (G.711, G.729, G723.1, Speex, iLBC e GSM).

De acordo com (FARIAS et al., 2008), OpenSIPStack tem seu núcleo baseado na biblioteca OPAL[3] e faz uso da biblioteca PWLib (Portable Windows Library) com o objetivo de tornar uma aplicação independente de sistema operacional. Além disso, em (FARIAS et al., 2008), a pilha OpenSIPStack tem sua documentação classificada como incompleta e insuficiente pelos autores do artigo, o que acarreta em alguns problemas em sua utilização.

3.2.4 Minisip

Minisip é uma biblioteca desenvolvida para construção de User Agents SIP voltados principalmente para aplicações VoIP. Escrita na linguagem C++, é desenvolvida por doutores e mestres de Royal Institute of Technology (KTH, Estocolmo, Suécia). Disponível através das licenças LGPL e GPL, apresenta suporte às plataformas Linux, Windows XP/2000 e Pocket PC.

Em suas principais características descritas em (MINISIP.ORG, 2006),se encaixam suporte a especificação (ROSENBERG et al., 2002), suporte a IM, suporte a videoconferência, suporte ao protocolo RTP e suporte a codec de áudio (G.711). Apesar disso, na sessão de donwloads[4] de sua *home page* é informado que nem todos os recursos estão disponíveis para a plataforma Windows XP e que a biblioteca ainda não foi completamente testada para operar neste sistema operacional.

[1]Disponível em <http://www.gnu.org/licenses/gpl.html>, acesso em novembro de 2010.
[2]Disponível em <http://www.mozilla.org/MPL/>, acesso em novembro de 2010.
[3]Disponível em <http://www.opalvoip.org/>, acesso em novembro de 2010.
[4]Disponível em <http://www.minisip.org/download.html>, acesso em novembro de 2010.

3.2.5 PJSIP

A PJSIP é uma pilha SIP escrita em C e mantida pelo grupo pjsip.org. Apresenta licença GPL[1], podendo ser liberada sob outras licenças, mediante acordo com os desenvolvedores (FARIAS et al., 2008). Das bibliotecas disponíveis para a implementação do protocolo SIP, a PJSIP é uma das que apresenta maior portabilidade para plataformas diferentes (Windows, Linux, MacOSX, RTEMS, Symbian OS, entre outros).

A PJSIP é composta por uma série de bibliotecas que visam dar suporte a recursos adicionais para o desenvolvimento de aplicativos, principalmente relacionados a VoIP. Nestes recursos estão disponíveis:

- Diversos codecs de áudio, entre eles: Speex, iLBC, GSM, G711, G722;
- Um conjunto de funções para tratamento e transporte de mídia;
- Um conjunto de funções específicas para desenvolvimento de agentes;
- Uma biblioteca para lidar com implementações que necessitam utilizar NAT;
- E por fim uma biblioteca que une as demais permitindo que essas cooperem entre si denominada PJSUA.

O grupo pjsip.org apresenta constantes atualizações em relação ao projeto PJSIP que, atualmente, está disponível em sua versão 1.8.5 (disponível a partir de 21 de outubro de 2010).

3.2.6 ReSIProcate

ReSIProcate é uma das mais famosas bibliotecas de implementação para o protocolo SIP. Escrita em C++ e desenvolvida pelo grupo resiprocate.org, apresenta licença de uso Vovida Software License[2]. Apresenta atualizações contínuas e suporte a vários sistemas operacionais, entre eles Windows e Linux.

A biblioteca ReSIProcate pode ser usada tanto para a construção de User Agents SIP quanto para Proxy Servers SIP. Apresenta uma interface denominada DUM (Dialog User Manager) que tem como objetivo simplificar a escrita de user agents, escondendo a complexidade do SIP. Apesar disso, não apresenta suporte a codecs e transmissão de mídia.

3.2.7 Sofia-SIP

A biblioteca Sofia-SIP é desenvolvida pelo grupo Nokia Research Center e tem como objetivo disponibilizar uma API para construção de User-Agent SIP. Escrita na linguagem C, suporta os sistemas operacionais GNU/Linux, MacOs e Windows. Desde agosto de 2005, Sofia-SIP foi disponibilizada sobre a licença LGLP e sua última versão está disponível desde dezembro de 2008.

Sofia-SIP disponibiliza três módulos distintos em sua implementação. Estes módulos visam simplificar a escrita de user-agents e estão divididos da seguinte forma:

- módulo nua: interface final para construção de clientes, disponibiliza funcionalidades para tratamento de chamadas, mensagens e eventos;

[1] Outros formatos de licença podem ser negociados com os desenvolvedores.

[2] Disponível em <http://www.resiprocate.org/License>, acesso em novembro de 2010.

- módulo su: módulo de portabilidade da biblioteca, contém a implementação de socket, timers e mecanismos de sincronização;
- módulo nta: módulo que disponibiliza uma interface simples para transações SIP, transporte e manipulação de mensagens.

3.3 Bibliotecas Escolhidas

Como descrito anteriormente, existem um conjunto grande de bibliotecas Open Source desenvolvidas e em desenvolvimento para implementar funcionalidades do protocolo SIP. Como não há espaço para uma comparação ampla entre todas elas, foram traçados alguns critérios para selecionar algumas bibliotecas para o estudo comparativo desenvolvido neste trabalho. Estes critérios se baseiam na parte prática do estudo, onde algumas funcionalidades do protocolo SIP serão adaptadas a um software já existente, a fim de disponibilizar uma sessão multimídia.

As bibliotecas PJSIP, ReSIProcate e Sofia-SIP foram consideradas as mais adequadas para o estudo desenvolvido neste trabalho. As três pilhas se encaixam nos requisitos propostos para o desenvolvimento prático, além de apresentarem características que tornem sua utilização mais simples em relação às outras pilhas analisadas.

4 METODOLOGIA

Para o desenvolvimento deste trabalho, será utilizada uma metodologia que avalie as características das bibliotecas comparadas. Porém, nem sempre, essas características estão presentes de forma quantitativa (tempo de execução, linhas de código necessárias para construção de uma aplicação, utilização de memória, entre outras) que produzam um resultado objetivo. Existem diversos outros fatores qualitativos referentes às bibliotecas, que mesmo produzindo resultados subjetivos, são importantes para determinar a qualidade do software utilizado.

As características de cada biblioteca analisada serão abordas de duas maneiras, a primeira utilizando métricas qualitativas e a segunda utilizando métricas quantitativas. Assim, essa sessão visa descrever os critérios para comparação das bibliotecas escolhidas e como serão abordadas durante o trabalho.

4.1 ISO/IEC 9126

Segundo a norma ISO/IEC 9126, descrita em (ISO/IEC, 2001), a especificação e avaliação de qualidade do produto de software são fatores chave para garantir qualidade adequada. Isto pode ser alcançado pela definição apropriada das características de qualidade, levando em conta consideração o uso pretendido do produto de software. É importante que cada característica relevante seja especificada e avaliada utilizando, quando possível, métricas validadas ou amplamente aceitas.

Além de analisar aspectos de qualidade de um software, a norma ISO/IEC 9126 define também seis grandes grupos de características: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

- funcionalidade: é a capacidade de o software satisfazer quaisquer funções adequando estados e necessidades implicadas quando usados sob condições específicas;
- confiabilidade: a capacidade de o software manter seu desempenho quando usado sob condições específicas;
- usabilidade: a capacidade de o software em ser fácil de usar e satisfazer o usuário, quando usado sob condições específicas;
- eficiência: os recursos usados por um software contido no sistema para alcançar o desempenho requerido sob condições específicas;
- manutenibilidade: os recursos necessários para fazer modificações específicas no software;

- portabilidade: capacidade de transferir o software para outros ambientes.

Segundo (STORCH, 2000) uma das grandes dificuldades da utilização desta norma é a especificação do método de avaliação, principalmente a definição das métricas aplicadas. Não são muitas as métricas de aceitação geral para as características descritas na norma, o que permite que grupos ou organizações de normalização estabeleçam seus próprios modelos de avaliação e métodos para a criação e validação de métricas relacionadas a estas características.

No caso do desenvolvimento deste trabalho, a figura do avaliador é definida pelo desenvolvedor que faz uso de bibliotecas de funções desenvolvidas por grupos diferentes para utilização em sua própria aplicação. Do ponto de vista de desenvolver, as principais características para comparar bibliotecas diferentes são referentes ao grupo usabilidade.

Os grupos funcionalidade, confiabilidade, manutenabilidade e portabilidade foram critérios utilizados para a escolha das bibliotecas analisadas, além disso, a eficiência (desempenho) das bibliotecas não será analisada, pois o software implementado necessita de interações com o usuário e os atrasos inseridos pelas diferentes implementações é desprezível em relação às necessidades ergonômicas humanas.

A norma ISO/IEC 9126 especifica critérios para a avaliação de qualidade de software. Este padrão define um número abrangente de critérios para a análise de qualidade de software em geral. O objetivo deste trabalho é realizar um estudo comparativo entre bibliotecas de funções tendo como foco principal critérios relacionados à usabilidade e requisitos importantes em relação ao desenvolvimento de aplicações utilizando estas bibliotecas. Portanto, mesmo existindo uma norma padronizada para avaliação de softwares em geral, esta não será aplicada durante o estudo apresentado. As sessões a seguir definem as métricas que serão utilizadas para avaliar e comparar cada uma das bibliotecas escolhidas.

4.2 Métricas Qualitativas

Segundo (SANT'ANNA, 2004), numa perspectiva de medição, qualidade de software deve ser definida em termos de atributos de produtos de software que são de interesse do usuário. Neste trabalho, o autor cita que a primeira tarefa de qualquer atividade de medição é identificar as entidades e atributos que se deseja medir, no entanto, muitas vezes é difícil definir os atributos de uma forma mensurável na qual todo mundo concorde. Todos querem construir e utilizar sistema de alta qualidade, mas nem sempre concordam com o significado de qualidade, o que gera dificuldades para medir qualidade de uma forma compreensível.

Muitos processos de medição começam medindo o que é conveniente ou fácil de medir, em vez de medir o que é realmente necessário. Normalmente, processos como estes falham, pois os dados resultantes não são úteis para quem desenvolve ou mantém o software (FENTON; PFLEEGER, 1998).

De acordo com muitos estudos sobre a aplicação de métricas e modelos de qualidade, um processo de medição, para ser efetivo, tem que ser focado em objetivos específicos, e sua interpretação deve ser baseada na caracterização e no entendimento desses objetivos (BASILI; CALDIERA; ROMBACH, 2002). A abordagem *Goal-Question-Metric* (GQM) se baseia nessa crença e foi definida originalmente por Basili(BASILI; CALDIERA; ROMBACH, 2002) para avaliar defeitos em uma série de projetos do Centro de Vôo Espacial da NASA. Depois foi aplicada em vários outros contextos.

O resultado da aplicação da metodologia *Goal-Question-Metric* é a especificação de um sistema de medição como alvo um conjunto de questões específicas e um conjunto de regras para a interpretação dos dados da medição. O modelo de avaliação resultante tem três passos:

- Listar os principais objetivos do processo de medição;
- Derivar, de cada objetivo, as perguntas que devem ser respondidas para determinar se os objetivos foram atingidos;
- Decidir o que precisa ser medido para ser capaz de responder as perguntas adequadamente.

Baseado na abordagem *Goal-Question-Metric* referentes a medida de qualidade de biblioteca analisadas neste trabalho, as principais características qualitativas que serão avaliadas são: Usabilidade, Documentação e Suporte. Os objetivos, perguntas e medidas que serão utilizadas na qualificação serão descritas nas subseções a seguir.

4.2.1 Usabilidade

O objetivo de analisar a Usabilidade de uma biblioteca é tentar definir qual a facilidade com que os desenvolvedores possam construir suas aplicações utilizando a entidade medida. Para isso, devem ser analisados fatores que contribuam diretamente para os seguintes requisitos:

- tempo necessário para o desenvolvimento de uma aplicação;
- a clareza com que as funções da API da biblioteca foram escritas;
- tempo necessário para instalação da biblioteca;
- facilidade para detectar possíveis erros de implementação.

Para qualificar a entidade medida será analisado seu código fonte e sua interface de utilização. Esta análise tenta responder uma série de perguntas que tem como objetivo classificar a biblioteca de acordo com a característica de usabilidade. Cada resposta a uma questão possui uma pontuação. Existem três tipos de respostas:

- Atende ao requisito: soma-se 2 pontos a nota para característica avaliada;
- Atende parcialmente ao requisito: soma-se 1 ponto a nota para característica avaliada;
- Não atende ao requisito: soma-se 0 pontos a nota para característica avaliada.

A partir da pontuação obtida pela biblioteca analisada será possível quantificar sua qualidade. A divisão entre a nota que a entidade medida receber e o máximo de pontos que seria possível atingir, resulta no percentual de qualidade que a biblioteca obteve de acordo com a característica avaliada.

A tabela 4.1 a seguir se refere às perguntas utilizadas para a métrica de Usabilidade.

Tabela 4.1: *Tabela de perguntas utilizada para avaliação de Usabilidade*

AVALIAÇÃO DE USABILIDADE	
PERGUNTAS	Nota
1. A biblioteca possui programa de instalação?	
2. O tempo necessário para sua instalação é satisfatório?	
3. Para a utilização da biblioteca não é necessária a instalação de softwares/bibliotecas adicionais além da IDE de programação?	
4. A biblioteca possui programas de exemplo?	
5. Os exemplos permitem compreender como a biblioteca funciona?	
6. Os exemplos de uso são facilmente acessíveis?	
7. O número de programas exemplo é satisfatório?	
8. A biblioteca apresenta programas de teste?	
9. O número de programas para teste de funcionalidades da biblioteca é satisfatório?	
10. A API da biblioteca faz uso de nomes satisfatórios para suas funções?	
11. A biblioteca disponibiliza a utilização de arquivos de log para facilitar a depuração de eventuais erros?	
NOTA MÁXIMA: 22	NOTA RECEBIDA:

4.2.2 Documentação

A Documentação descreve cada parte do código fonte, geralmente uma função, uma classe, um simples trecho ou módulo. Consiste também no conjunto de manuais gerais e técnicos, além de diagramas explicando o funcionamento como um todo ou cada parte dele.

O objetivo de analisar a Documentação de uma biblioteca é tentar definir quão auto explicativa é a entidade medida. Para qualificar o critério de Documentação será analisado o código fonte e o material disponível no website de cada biblioteca. Esta análise tenta responder uma série de perguntas que tem como objetivo classificar a biblioteca de acordo com a característica de Documentação. Cada resposta a uma questão possui uma pontuação e os tipos de respostas são os mesmos definidos na subseção anterior.

A partir da pontuação obtida pela biblioteca analisada será possível quantificar sua qualidade. A divisão entre a nota que a entidade medida receber e o máximo de pontos que seria possível atingir, resulta no percentual de qualidade que a biblioteca obteve de acordo com a característica avaliada. A tabela 4.2 a seguir se refere às perguntas utilizadas para a métrica de Documentação.

4.2.3 Suporte

A análise do nível de Suporte prestado pelo grupo que desenvolve uma biblioteca é importante, pois, possíveis dúvidas que desenvolvedores terão durante o desenvolvimento de suas aplicações precisam ser respondidas. Se dúvidas quanto ao uso de uma biblioteca não podem ser sanadas, fica inviável a utilização desta ferramenta.

Tabela 4.2: *Tabela de perguntas utilizada para avaliação de Documentação*

AVALIAÇÃO DE DOCUMENTAÇÃO	
PERGUNTAS	Nota
1. A biblioteca apresenta manual de instalação?	
2. A biblioteca faz uso de alguma ferramenta ou padrão para documentação?	
3. A biblioteca possui um tutorial de boa qualidade?	
4. Todas as funções da biblioteca apresentam documentação?	
5. Os arquivos de código fonte seguem o mesmo padrão de documentação?	
6. Os exemplos de uso da biblioteca apresentam documentação?	
7. A documentação dos exemplos de uso é suficiente para o entendimento dos mesmos?	
8. A documentação geral da biblioteca é suficiente para que o desenvolvedor possa utilizá-la de maneira adequada?	
NOTA MÁXIMA: 16	NOTA RECEBIDA:

O Suporte será analisado a partir das ferramentas que cada grupo de desenvolvimento das bibliotecas analisadas disponibiliza em sua *home page*. Esta análise segue o mesmo padrão das características descritas nas subseções anteriores. Consiste em um questionário com o objetivo de obter uma pontuação para quantificar as entidades medidas. A tabela 4.3 a seguir se refere às perguntas utilizadas para a métrica de Suporte.

Tabela 4.3: *Tabela de perguntas utilizada para avaliação de Suporte*

AVALIAÇÃO DE SUPORTE	
PERGUNTAS	Nota
1. O grupo que desenvolve a biblioteca disponibiliza um fórum de discussão online?	
2. O grupo que desenvolve a biblioteca disponibiliza uma lista de emails para publicação de informações referentes a seu desenvolvimento ou discussões sobre o projeto?	
3. A página principal do projeto da biblioteca apresenta lista de FAQ?	
4. A página principal do projeto da biblioteca é objetiva e de fácil acesso?	
NOTA MÁXIMA: 8	NOTA RECEBIDA:

4.3 Métricas Quantitativas

Existem diversas formas de se analisar objetivamente uma aplicação. Embora, muitas vezes, essas formas sejam estatísticas, em geral elas oferecem uma boa visão sobre o uso de um determinado software em um dado contexto. Assim, em um primeiro momento, é

feita uma análise sobre os aspectos quantitativos do código escrito para avaliar o esforço necessário ao se escrever uma aplicação fazendo uso de uma determinada biblioteca de funções. Pela necessidade de interação com o usuário final, a análise de desempenho do software implementado não será realizada neste trabalho.

4.3.1 Métricas de código

A análise de um código fonte em geral está relacionada com a capacidade do programador em criar estruturas adequadas para o problema. Contudo, como este trabalho visa analisar comparativamente implementações que dependem fortemente das APIs disponibilizadas pelas bibliotecas, não há uma grande flexibilidade em termos de se utilizar outras estruturas se não as fornecidas pelas próprias APIs.

Embora seja discutível a utilização de métricas de código para qualificar um software que utiliza módulos escritos em linguagens diferentes, uma diferença muito grande em relação número de linhas de código entre programas similares podem ser indicadores claros de complexidade e esforço. Existem basicamente três tipos de métricas para linhas de código.

- LOC ou Lines of Code: considera todas as linhas de código presentes, desconsiderando linhas em branco e comentários;
- eLOC ou Effective Lines of Code: desconsidera comentários, linhas em branco e parênteses;
- lLOC ou Logical Lines of Code: considera apenas declarações de código terminadas por ponto e vírgula.

A figura 4.1 exemplifica as métricas acima listadas.

```
Source code line       LOC   eLOC  lLOC  Comment  Blank
-------------------------------------------------------
if (x<10)   // test    x     x           x
{                      x
  // update y                           x
                                                  x
  y = x + 1;           x     x     x
}                      x
-------------------------------------------------------
```

Figura 4.1: Métricas LOC, eLOc e lLOC.

A análise de linhas de código será utilizada com o intuito de qualificar as implementações que utilizam bibliotecas de funções diferentes. Assim, é possível estimar a complexidade e esforço gerados a partir de cada implementação, e traçar um comparativo para estas características.

5 IMPLEMENTAÇÃO

O objetivo específico deste trabalho consiste na adaptação de algumas funcionalidades do protocolo SIP a um software para videoconferência já existente. A descrição do software e suas funcionalidades serão apresentadas neste capítulo. Além disso, será apresentado como cada biblioteca escolhida foi adaptada ao software base, e os resultados obtidos a partir da adaptação. A implementação visa auxiliar a análise qualitativa e quantitativa das bibliotecas e produzir resultados que possam interpretar o esforço necessário para criação de um cliente SIP.

5.1 Descrição do Aplicativo

O Software *Apresentador* é desenvolvido pelo laboratório de Projetos em Áudio e Vídeo (PRAV) da Universidade Federal do Rio Grande do Sul. Ele faz parte de uma solução cujo objetivo é a criação de um ambiente completo em software e hardware para transmissão de multimídia interativa em uma infraestrutura para Ensino a Distância (EAD).

O *Apresentador* é um software cliente utilizado para transmissão e recepção de áudio e vídeo de alta qualidade. Ele apresenta um sistema para codificação e decodificação de áudio e vídeo, disponibilizando áudio codificado através do codec MP2[1] e vídeo codificado através do codec MPEG-4[2]. A transmissão de dados multimídia é realizada através de um protocolo próprio, não sendo compatível com os protocolos padronizados, como por exemplo, o RTP.

Uma variação do software *Apresentador* será utilizada como base para a implementação prevista neste trabalho. O que se pretende é utilizar uma solução já desenvolvida e agregar a ela algumas funcionalidades disponibilizadas pelo protocolo SIP a fim de permitir que o software gerado possa realizar uma sessão multimídia baseada no padrão SIP.

5.2 Implementação Proposta

Para o transporte de mídia em tempo real, se fez necessária a implementação do protocolo RTP junto ao software base. Para isso, foi utilizada a biblioteca JRTPLIB[3], desenvolvida por Jori Liesenborgs. A JRTPLIB é uma biblioteca orientada a objetos e escrita em C++ que disponibiliza a implementação do protocolo RTP e RTCP. A partir de então, os requisitos necessários para adaptação de funcionalidades do protocolo SIP foram supridos, bastando apenas programar um módulo que faz uso das bibliotecas SIP analisadas.

[1] MP2: codec de áudio MPEG-1 Audio Layer II

[2] MPEG-4: MPEG-4 Part 2 ou MPEG-4 Visual

[3] JRTPLIB, disponível em <http://research.edm.uhasselt.be/ jori/page/index.php?n=CS.Jrtplib>, acesso em novembro de 2010.

Como base na documentação e nos exemplos presentes em cada biblioteca, se espera criar um *User Agent* SIP e posteriormente integrá-lo ao software base. A partir de então o software base passará a disponibilizar a criação, estabilização e finalização de sessões com base no protocolo SIP.

Para iniciar uma sessão, um cliente SIP chama um outro ponto final SIP enviando uma mensagem com o método *INVITE*. As capacidades de mídia que o ponto de origem da chamada pode receber e enviar, além do endereço de transporte onde ele espera que o ponto de destino da chamada envie esses dados de mídia são enviados junto com o pedido de convite. O ponto de destino da chamada precisa indicar que ele está aceitando o pedido. Esse é o objetivo da resposta *OK*. Uma vez que o pedido foi um convite, a resposta *OK* também contém as capacidades de mídia do ponto de destino da chamada e onde ele está esperando receber os dados de mídia. O cliente que iniciou a chamada precisa confirmar que recebeu de maneira adequada a resposta do ponto de destino e faz uso do método *ACK* para indicar o sucesso. A partir deste ponto a sessão está inicializada e a troca de mídia pode ser efetuada. Por fim, um dos pontos pode terminar a sessão, enviando o método *BYE* para indicar o fim de sessão. Como resposta recebe *OK* indicando que a outra parte foi avisada sobre o término. A figura 5.1 demonstra o diagrama de mensagens SIP trocadas entre dois *user agents* que se espera obter ao fim da implementação.

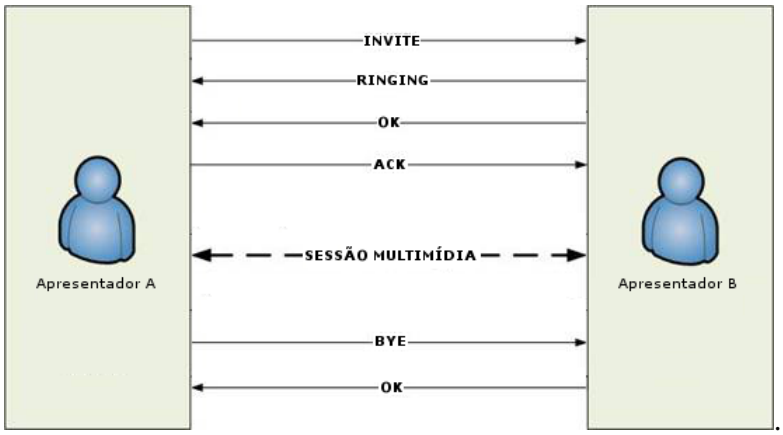

Figura 5.1: Troca de mensagens esperada após implementação

5.3 Adaptação com PJSIP

Para a implementação sugerida foi preterida apenas a utilização de funções referentes a sinalização SIP presentes na biblioteca PJSIP. Apesar disso, encontrou-se dificuldades em separar o módulo de mídia da biblioteca (PJMEDIA) da parte referente a sinalização. Por isso, se fez necessária a utilização de algumas funções da PJMEDIA para criação e manipulação do corpo da mensagem de descrição de sessão (SDP).

O primeiro passo para a adaptação foi a criação de uma *thread* que contém as funções de inicialização da biblioteca e um laço para gerenciar eventos ocorridos durante uma sessão. Esta *thread* garante que o programa criado para sinalização execute concorrentemente com os outros elementos do software. O código a seguir, ilustra os comandos utilizados para a inicialização dos recursos da biblioteca e o laço de espera para eventos de uma sessão. Declarações de variáveis e tratamento de erros foram omitidos para fins de clareza.

```
void *pjsipMainThread(void *parameters)
{
  //...

  // Inicializa PJ Library
  status = pj_init();

  // Inicializa PJLIB-UTIL, biblioteca auxiliar
  status = pjlib_util_init();

  // Inicializa recursos para manipulação de memória
  pj_caching_pool_init(&cp, &pj_pool_factory_default_policy, 0);

  //...

  // Cria endpoint
  status = pjsip_endpt_create(&cp.factory, strEndName, &myEndPoint);

  //...

  // Inicializa transporte de mensagens sobre sinalização UDP
  status = pjsip_udp_transport_start(myEndPoint, &addr.ipv4, 0, 1, 0);

  // Cria e inicializa hash table de transações
  status = pjsip_tsx_layer_init_module(myEndPoint);

  // Cria e inicializa módulo do user agent
  status = pjsip_ua_init_module(myEndPoint, NULL);

  // Inicializa callbacks para manipulação de eventos
  {
    // Seta callback chamado quando o estado de uma ligação é alterado
    inv_cb.on_state_changed = &call_on_state_changed;

    // Seta callback chamado quando uma nova sessão é criada
    inv_cb.on_new_session = &call_on_forked;

    // Seta callback chamado quando negociação de mídia foi concluída
    inv_cb.on_media_update = &call_on_media_update;

    // Inicializa módulo de sessão
    status = pjsip_inv_usage_init(myEndPoint, &inv_cb);
  }

  // Registra módulo para receber mensagens
  status = pjsip_endpt_register_module(myEndPoint, &modUserAgent);

  // Laço para tratamento de eventos
  while (sipThreadRunning)
  {
    // manipulação do estado de uma chamada
    // ...

    pjsip_endpt_handle_events(myEndPoint, &timeout);
  }

  //...
```

```
58 }
```

Listagem 5.1: Inicialização da biblioteca PJSIP.

Depois de inicializada, a aplicação aguarda a interação do usuário com o software a fim de realizar, aceitar, rejeitar ou finalizar uma chamada. A figura 5.2 ilustra a estrutura lógica utilizada pela aplicação para interação com o usuário.

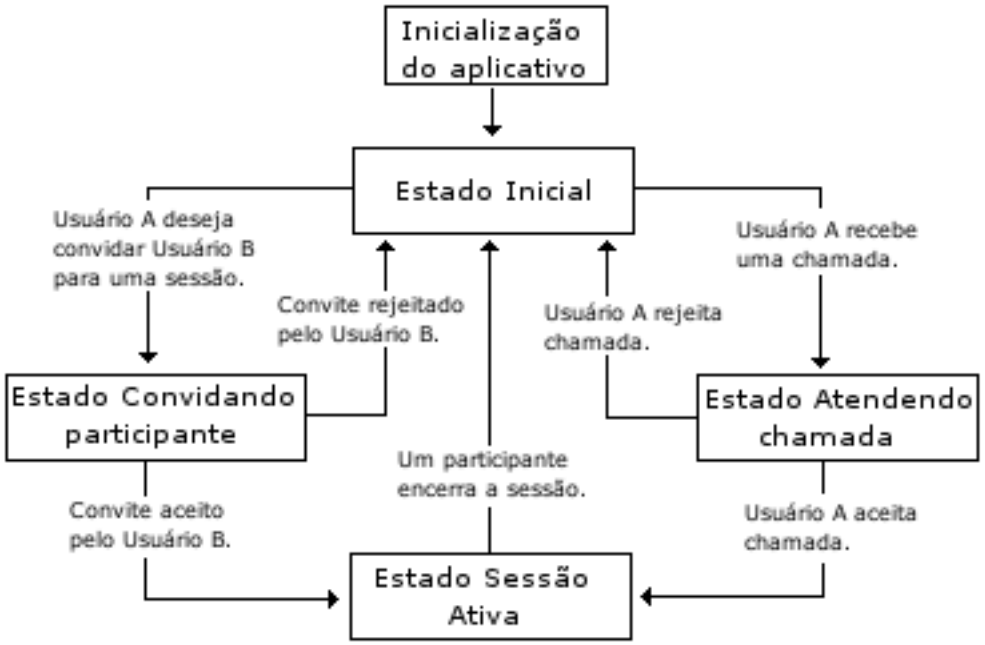

Figura 5.2: Estrutura lógica utilizada pela aplicação

Os estados descritos na figura 5.2 são alterados através da troca de mensagens SIP efetuada através da utilização da biblioteca. Como descrito anteriormente, o protocolo SIP funciona através de envio de métodos e respostas. Os códigos a seguir, ilustram como é realizado o envio de métodos e respostas utilizando a API disponível pela biblioteca PJSIP.

```
1  // Função invocada sempre que ocorre a requisição de uma resposta
2  static pj_bool_t on_rx_request(pjsip_rx_data *rdata)
3  {
4    // ...
5
6    switch (rdata->msg_info.msg->line.req.method.id)
7    {
8    case PJSIP_INVITE_METHOD:
9      // Verifica se a requisição pode ser atendida
10     status = pjsip_inv_verify_request(rdata, &options, NULL, NULL,
           myEndPoint, NULL);
11     //...
12
13     // Inicializa user agente server(UAS) com informações recebidas
14     status = pjsip_dlg_create_uas( pjsip_ua_instance(), rdata, &
           local_uri, &dlg);
15     //...
```

```
16
17      // Cria SDP local
18      status = createLocalSdp(&local_sdp, dlg->pool, 2);
19
20      // Cria sessão, passando UAS dialog e capacidades SDP
21      status = pjsip_inv_create_uas(dlg, rdata, local_sdp, 0, &
            myInviteSession);
22
23      // Cria resposta inicial, 180 Ringing
24      status = pjsip_inv_initial_answer(myInviteSession, rdata, 180, NULL
            , NULL, &tdata);
25
26      // Envia resposta 180 Ringing.
27      status = pjsip_inv_send_msg(myInviteSession, tdata);
28
29      // Aguarda usuário atender ou rejeitar ligação
30      //...
31
32      // Cria resposta
33      status = pjsip_inv_answer(myInviteSession, reply, NULL, NULL, &
            tdata);
34
35      // Envia resposta
36      status = pjsip_inv_send_msg(g_inv, tdata);
37
38      //...
39      break;
40
41    case PJSIP_BYE_METHOD:
42      // Cria resposta referente a BYE recebido
43      //...
44
45      // Envio de resposta
46      pjsip_endpt_respond_stateless(myEndPoint, rdata, reply, &reason,
            NULL, NULL);
47      break;
48
49    // Outros métodos
50    //...
51
52    case PJSIP_OTHER_METHOD:
53      // Cria resposta referente a métodos desconhecidos
54      //...
55
56      pjsip_endpt_respond_stateless(myEndPoint, rdata, reply, &reason,
            NULL, NULL);
57      break;
58    }
59
60    //...
61
62    return PJ_TRUE;
63  }
```

Listagem 5.2: Envio de resposta utilizando biblioteca PJSIP.

```
1  // Função invocada para envio de método INVITE
2  int sendInviteMethod(pjsip_inv_session* myInviteSession)
```

```
3  {
4    //...
5
6    // Cria user agente client(UAC)
7    status = pjsip_dlg_create_uac(pjsip_ua_instance(), &local_uri, &
         local_uri, &dst_uri, &dst_uri, &dlg);
8
9    // Cria SDP local
10   status = createLocalSdp(&local_sdp, dlg->pool, 2);
11
12   // Cria sessão, passando UAC dialog e capacidades SDP
13   status = pjsip_inv_create_uac(dlg, local_sdp, 0, &myInviteSession);
14
15   // Cria requisição inicial para INVITE
16   status = pjsip_inv_invite(myInviteSession, &tdata);
17
18   // Envia requisição inicial para INVITE
19   status = pjsip_inv_send_msg(myInviteSession, tdata);
20
21   //...
22
23   return 0;
24 }
25
26 // Função invocada para envio de método BYE
27 int sendByeMethod(pjsip_inv_session* myInviteSession)
28 {
29   //...
30
31   status = pjsip_inv_end_session(myInviteSession, statusCode, NULL, &
         tdata);
32
33   status = pjsip_inv_send_msg(myInviteSession, tdata);
34   //...
35
36   return 0;
37 }
```

Listagem 5.3: Envio de métodos utilizando biblioteca PJSIP.

A implementação do módulo de sinalização utilizando a biblioteca PJSIP se mostrou extensa e complexa em relação às outras bibliotecas analisadas. A API disponibilizada pela biblioteca requer o uso de tipos próprios definidos no projeto, o que torna a programação menos flexível e diminui a legibilidade do código criado. Mesmo para o desenvolvimento de uma aplicativo simples, a compreensão dos exemplos disponíveis no projeto da biblioteca são necessários, sem estes, a criação de uma aplicação seria extremamente complicada.

5.4 Adaptação com ReSIProcate

A biblioteca ReSIProcate disponibiliza uma API para criação de *User Agents*, *Proxy Servers*, *Registrar Server* entre outros elementos presentes na arquitetura SIP. Para facilitar a criação de *User Agents* a API dispõe de um módulo denominado DUM (Dialog User Manager). Utilizando como base este módulo foi desenvolvido um *User Agent* capaz de realizar a sinalização SIP proposta anteriormente. Feito isso, a sinalização foi adaptada

ao software base utilizado.

Um dos fatores determinantes para que esta pilha se tornasse uma das mais famosas atualmente, é sua simplicidade em relação às outras bibliotecas disponíveis. O tempo necessário para se criar uma aplicação utilizando ReSIProcate é bem menor do que o tempo gasto com a implementação de um aplicativo parecido utilizando as bibliotecas Sofia-SIP ou PJSIP por exemplo. Este fato pode ser analisado comparando o número de linhas de código necessárias para a escrita de uma aplicação com funcionalidades semelhantes.

O primeiro passo para a adaptação foi a criação de uma *thread* que contém a criação dos objetos necessários para a inicialização da biblioteca e um laço para gerenciar a troca de mensagens durante uma sessão. A *thread* visa disponibilizar uma execução paralela entre a sinalização SIP e o software que interage com o usuário. A interação entre o software e a sinalização segue a mesma estrutura lógica apresentada na sessão anterior e ilustrada pela figura 5.2. O código a seguir, ilustra o objetos instanciados para a inicialização dos recursos da biblioteca e o laço que gerencia as ações do usuário. Para fins de clareza, foram omitidos algumas declarações de objetos e tratamentos de erros.

```
1  void *resiprocateMainThread(void *parameter)
2  {
3    //...
4
5    // Passo inicial de interação entre módulo de aplicação e pilha sip
6    SipStack stackUA;
7
8    // Cria interface DUM para manipulação da pilha criada
9    DialogUsageManager* dumUA = new DialogUsageManager(stackUA);
10
11   // Inicializa módulo de transporte
12   dumUA->addTransport(UDP, RESIP_SIP_PORT);
13
14   // Cria profile para User Agent instanciados
15   SharedPtr<MasterProfile> uaMasterProfile(new MasterProfile);
16   dumUA->setMasterProfile(uaMasterProfile);
17
18   // Instancia User Agent
19   MyClassForUA ua;
20
21   // Atrela User Agent com a interface DUM
22   dumUA->setInviteSessionHandler(&ua);
23   dumUA->setClientRegistrationHandler(&ua);
24
25   // Inicializa profile com endereço sip local
26   dumUA->getMasterProfile()->setDefaultFrom(mySipAddr);
27
28   //...
29
30   // laço para envio de métodos e respostas
31   while (resipThreadRunning)
32   {
33     if (!(ua.done))
34     {
35       // Envia método INVITE
36       if (callStatus == SIP_STATUS_INVITING)
37         dumUA->send(dumUA->makeInviteSession(destSipAddr, ua.mSdp));
38
39       // Aceita chamada pendente
```

```
40          if (callStatus == SIP_STATUS_ACCEPTING_CALL)
41            ua.acceptCall();
42
43          // Rejeita chamada pendente
44          if (callStatus == SIP_STATUS_REJECTING_CALL)
45            ua.rejectCall();
46
47          // Envia método BYE
48          if (callStatus == SIP_STATUS_QUITTING)
49            ua.hangupCall();
50        }
51
52        //...
53      }
54
55      delete dumUA;
56
57      //...
58  }
```

Listagem 5.4: Inicialização da biblioteca ReSIProcate.

O objeto que implementa um *User Agent* é utilizado para inicialização da biblioteca. Este objeto foi criado a partir de duas classes que definem métodos herdados da classe *InviteSessionHandler* disponível a partir do módulo DUM. Dentro deste objeto são declarados métodos invocados quando o estado de uma sessão é alterado. A classe *MyInviteSessionHandler* define métodos de *callback* para eventos específicos herdados da classe *InviteSessionHandler*, enquanto a classe *MyClassForUA* implementa um *User Agent* que herda todas as funcionalidades da classe *MyInviteSessionHandler*, além de definir métodos específicos a fim de responder chamadas e enviar os métodos SIP *INVITE* e *BYE*. O código criado para definição destas classes é ilustrado a seguir.

```
1   class MyInviteSessionHandler : public InviteSessionHandler
2   {
3     //...
4
5     // método invocado quando UA envia INVITE
6     virtual void onNewSession(ClientInviteSessionHandle sis,
          InviteSession::OfferAnswerType oat, const SipMessage& msg)
7     {
8       mode = _RESIP_CLIENT_MODE;
9       mClientSis = sis;
10    }
11
12    // método invocado quando UA recebe INVITE
13    virtual void onNewSession(ServerInviteSessionHandle sis,
          InviteSession::OfferAnswerType oat, const SipMessage& msg)
14    {
15      mode = _RESIP_SERVER_MODE;
16      mServerSis = sis;
17
18      // Envia resposta 180 Ringing
19      mServerSis->provisional(180);
20    }
21
22    // método invocado quando recebe oferta remota para SDP
23    virtual void onOffer(InviteSessionHandle is, const SipMessage& msg,
          const SdpContents& sdp)
```

```
24    {
25      // Invoca método que providencia resposta ao SDP(negociação)
26      is->provideAnswer(sdp);
27
28      //...
29    }
30
31    // Outros métodos invocados em eventos específicos
32    //...
33  };
```

Listagem 5.5: Classe MyInviteSessionHandler.

```
1   class MyClassForUA : public MyInviteSessionHandler
2   {
3     //...
4
5     // Método utilizado para rejeitar uma chamada
6     void rejectCall()
7     {
8       //...
9
10      if(mode == _RESIP_SERVER_MODE)
11      {
12        mServerSis->reject(code);
13        //...
14      }
15    }
16
17    // Método utilizado para aceitar uma chamada
18    void acceptCall()
19    {
20      if(mode == _RESIP_SERVER_MODE)
21      {
22        mServerSis->accept();
23        //...
24      }
25    }
26
27    // Método utilizado para terminar uma chamada
28    void hangupCall()
29    {
30      if (mode == _RESIP_CLIENT_MODE)
31      {
32        mClientSis->end();
33        done = true;
34      }
35      else if(mode == _RESIP_SERVER_MODE)
36      {
37        mServerSis->end();
38        done = true;
39      }
40    }
41
42    // Método invocado quando uma sessão foi estabilizada
43    virtual void onConnected(ClientInviteSessionHandle is, const
          SipMessage& msg)
44      {
```

```
45      // Sessão ativa
46      //...
47    }
48  };
```

Listagem 5.6: Classe MyClassForUA.

A API disponibilizada pela biblioteca ReSIProcate se mostrou bastante simples em relação a criação de um *User Agent*. Um ponto fraco em relação a sua utilização é que os exemplos existentes no projeto da pilha implementam aplicativos simples e nenhum exemplo foi encontrado para criação de aplicativos com mais funcionalidades. Um exemplo mais sofisticado se faz necessário, pois a biblioteca não apresenta uma documentação muito boa de seu código, gerando certas dúvidas quando se faz necessário o uso de recursos não exemplificado.

5.5 Adaptação com Sofia-SIP

Sofia-SIP é uma biblioteca que disponibiliza funções específicas para a criação de um *User Agent*. Assim, não foram encontrados problemas para separar módulos específicos da biblioteca, já que seu projeto foca apenas a parte de sinalização do protocolo SIP. Para a adaptação ao software base, foi empregada a mesma estrutura lógica ilustrada na figura 5.2 e utilizada nas outras adaptações.

Para criação da sinalização foi utilizada as interfaces *nua* e *su* presentes na biblioteca. O módulo *su* implementa funções para criação *sockets*, *timers* e mecanismos sincronização com a objetivo de prover portabilidade entre plataformas diferentes. Já o módulo *nua* dispõe de funções de mais alto nível para tratar chamadas, mensagens e eventos.

O primeiro passo foi a criação de uma *thread* que contém as funções de inicialização da biblioteca e um laço para gerenciar eventos ocorridos durante uma sessão. Como nas adaptações citadas anteriormente, o objetivo desta *thread* é prover uma execução concorrente entre a sinalização SIP e a aplicação que interage com o usuário. Os códigos a seguir, ilustram os comandos utilizados para a inicialização dos recursos da biblioteca e o laço que gerencia os estados de uma sessão.

```
1  void *sofiaMainThread(void *parameters)
2  {
3    //...
4
5    // Inicializa Sofia-SIP library
6    su_init ();
7
8    // Inicializa estrutura su_home_t
9    su_home_init(myEndPoint->home);
10
11   //...
12
13   // Cria SDP local
14   local_sdp = buildLocalSdp(sdpConfig);
15
16   // Cria User Agent e o inicializa de acordo com TAGS desejadas
17   myEndPoint->nua_object = nua_create( myEndPoint->root,
18                                        event_callback,
19                                        myEndPoint,
20                                        NUTAG_URL(mySipAddr),
21                                        NUTAG_MEDIA_ENABLE(1),
```

```
22                                                  NUTAG_MEDIA_ADDRESS(myEndPoint->
                                                        media),
23                                                  NUTAG_SOA_NAME("default"),
24                                                  SOATAG_USER_SDP_STR(local_sdp),
25                                                  SOATAG_AF(SOA_AF_IP4_ONLY),
26                                                  TAG_END());
27
28    // Seta parâmetros adicionais ao User Agent criado
29    nua_set_params( myEndPoint->nua_object, NUTAG_ENABLEINVITE(1),
          TAG_NULL());
30
31    // Cria thread para tratamento de eventos
32    ret = pthread_create(&handleEventsThread_id, NULL, sofiaEventThread,
          myEndPoint);
33
34    // Loop que suspende a execução desta thread
35    su_root_run(myEndPoint->root);
36
37    //...
38  }
```

Listagem 5.7: Inicialização da biblioteca Sofia-SIP.

```
1   // Função executado como uma thread para tratamento de eventos
2   void *sofiaEventThread(void *param)
3   {
4     //...
5
6     // Laço para tratamento de eventos
7     while(eventThreadRunning)
8     {
9       // De acordo com o estado da aplicação,
10      // Envia INVITE, BYE ou responde convite para sessão
11      {
12        if (callState == SIP_STATUS_INVITING)
13        {
14          //...
15          ret = sendInviteMethod(myEndPoint, destinationAddr);
16        }
17
18        if (callState == SIP_STATUS_RESPONDING_INCOMING_CALL)
19        {
20          // ...
21          answerRequest(myEndPoint, reply, phrase);
22        }
23
24        if (SIP_STATUS_QUITTING)
25        {
26          //...
27          int sendByeMethod(myEndPoint);
28        }
29      }
30
31      Sleep(SOFIA_EVENT_SLEEP_TIME);
32    }
33
34  }
```

Listagem 5.8: Laço para envio de métodos e respostas SIP.

Ao se criar um *user agent* através da interface *nua*, é necessária a especificação de uma função de *callback*. Esta função é chamada sempre que um novo evento acontece. Um evento pode ser interpretado com um método recebido, uma resposta recebida ou indicar que o estado de uma sessão foi alterado.

A função de *callback* faz o papel de servidor dentro de um *user agent*, respondendo requisições e alterando o estado de uma sessão. Um fato relevante a ser mencionado é que nem todas as requisições necessitam ter suas respostas implementadas pelo programador. Algumas delas são geradas automaticamente pela API, como por exemplo, as respostas *100 Trying* e *180 Ringing*. Outras respostas como *200 OK* devem ser encaminhadas pelo programador já que necessitam da interação do usuário para serem definidas.

Para toda a requisição que não é respondida automaticamente pela API, é necessária a criação de uma operação indicando que existe uma solicitação pendente. Esta solicitação deve ser respondida posteriormente para que os estados de uma sessão possam ser alterados. O código listado a seguir ilustra como a função de *callback* é utilizada. Declarações de variáveis, tratamento de erros e algumas chamadas foram omitidos para fins de clareza.

```
1  void event_callback( nua_event_t  event,
2                       int          status,
3                       char const   *phrase,
4                       nua_t        *nua,
5                       nua_magic_t  *magic,
6                       nua_handle_t *nh,
7                       nua_hmagic_t *hmagic,
8                       sip_t const  *sip,
9                       tagi_t       tags[])
10 {
11   //...
12
13   switch (event)
14   {
15   case nua_r_invite:
16     // Recebeu resposta referente a um INVITE enviado
17     {
18       // Trata resposta recebida
19       //...
20     }
21     break;
22
23   case nua_i_invite:
24     // Recebeu um pedido de chamada
25     {
26       // Cria operação para tratar pedido, e a associa ao endPoint
27       {
28         //...
29       }
30
31       // Associa um callback para operação criada
32       nua_handle_bind(op->handle = nh, op);
33     }
34     break;
35
36   case nua_i_active:
37     // Evento para chamada ativa
```

```
38      {
39        // negociação SDP efetuada, sinaliza aplicação para envio de
              mídia
40        // ...
41      }
42      break;
43
44    case nua_i_bye:
45      // Recebeu método BYE
46      {
47        // Resposta é enviada automaticamente pela API
48      }
49      break;
50
51    case nua_i_terminated:
52      // Evento indicando que sessão terminou
53      {
54        // Finaliza thread de eventos
55        eventThreadRunning = false;
56      }
57      break;
58
59    // Outras requisições, resposta ou eventos recebidas
60    // ...
61
62    }
63
64    //...
65  }
```

Listagem 5.9: Função definida para recebimento de eventos.

As funções listadas a seguir ilustram como os comandos da biblioteca Sofia-SIP foram utilizados para o envio de métodos e respostas a requisições.

```
1  // Função invocada para envio de método INVITE
2  int sendInviteMethod(end_point_t *myEndPoint, char *destSipAddr)
3  {
4    // Aloca e inicializa nova estrutura de header
5    to = sip_to_make(myEndPoint->home, destSipAddr);
6    //...
7
8    // Aloca memória para nova operação
9    op = (end_oper_t *)su_zalloc(myEndPoint->home, sizeof(*op));
10
11   //...
12
13   // Aloca handle para nova operação e associa TAGS desejadas
14   op->handle = nua_handle(myEndPoint->nua_object, op, SIPTAG_TO(to),
         TAG_END());
15
16   // Converte estrutura de header para string
17   op->op_ident = sip_header_as_string(myEndPoint->home, (sip_header_t
         *)to);
18
19   // Seta método utilizado
20   op->op_method = SIP_METHOD_INVITE;
21
22   // Envia método INVITE ao destino
```

```
23    nua_invite(op->handle, TAG_END());
24
25    return 0;
26  }
27
28  // Função invocada para envio de método BYE
29  int sendByeMethod(end_point_t *myEndPoint)
30  {
31    // Procura por ligação ativa
32    handle = oper_find_call(myEndPoint);
33
34    //...
35
36    // Envia method Bye
37    nua_bye(handle, TAG_END());
38
39    return 0;
40  }
```

Listagem 5.10: Envio de métodos utilizando biblioteca Sofia-SIP.

```
1  // Função desenvolvida para responder requisições
2  void answerRequest(end_point_t *endpt, int replyValue, char const *
       phrase)
3  {
4    //...
5
6    // função criada para varrer lista
7    // retorna requisição não respondida
8    if((op = oper_find_unanswered(endpt)))
9    {
10     nua_respond(op->handle, replyValue, phrase, TAG_END());
11   }
12
13   //...
14 }
```

Listagem 5.11: Envio de resposta utilizando biblioteca Sofia-SIP.

A API disponibilizada pela biblioteca Sofia-SIP se mostrou bastante flexível. A liberdade proporcionada ao programador por outro lado pode tornar a criação de aplicativos um pouco complexa, visto que é necessária a criação de um código extenso para se fazer uso desta biblioteca. A principal dificuldade encontrada para a implementação de uma sinalização SIP utilizado a biblioteca Sofia-SIP foi a falta de exemplos no projeto da biblioteca. Foi necessário baixar versões de projetos anteriores a utilizada no estudo para ter acesso a algum tipo de exemplo de uso. Por outro lado, o tutorial existente na página principal da biblioteca oferece um guia de boa qualidade, proporcionando aos programadores com pouca experiência com esta pilha uma documentação.

5.6 Demostração dos resultados

A figura 5.3 apresenta a troca de mensagens obtidas após uma sessão utilizando a adaptação proposta. A figura demonstra a troca de mensagens entre um *User Agents* desenvolvido a partir da biblioteca PJSIP e outro a partir da biblioteca Sofia-SIP. Para

captura dos dados de rede foi utilizado o software Wireshark[1] versão 1.4.2. Para fins de clareza, apenas a troca de mensagens SIP foi capturada.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.163	192.168.0.132	SIP/SDP	Request: INVITE sip:192.168.0.132, with session description
4	0.016420	192.168.0.132	192.168.0.163	SIP	Status: 180 Ringing
5	3.580359	192.168.0.132	192.168.0.163	SIP/SDP	Status: 200 OK, with session description
6	3.581045	192.168.0.163	192.168.0.132	SIP	Request: ACK sip:simpleuas@192.168.0.132:5060
9	9.513923	192.168.0.132	192.168.0.163	SIP	Request: BYE sip:192.168.0.163
10	9.514306	192.168.0.163	192.168.0.132	SIP	Status: 200 OK

Figura 5.3: Troca de mensagens SIP obtidas.

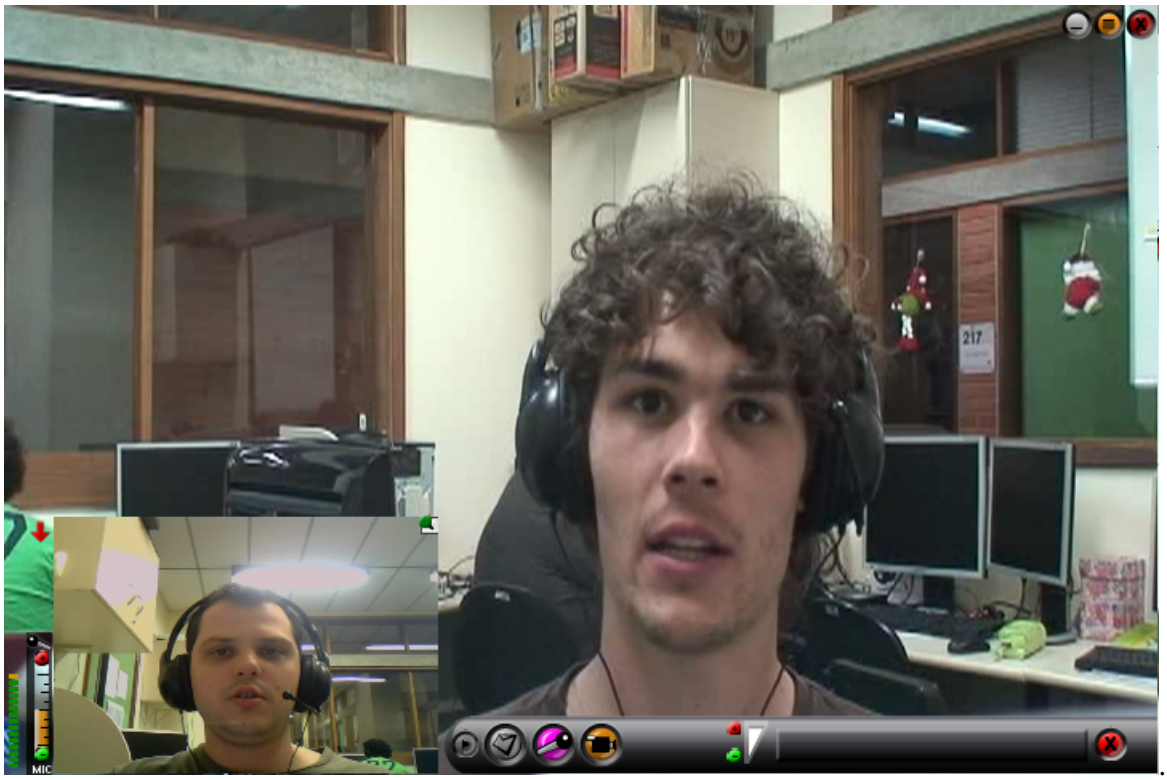

Figura 5.4: Interface do software com um sessão em andamento.

[1] disponível em http://www.wireshark.org/download.html, acesso em novembro de 2010.

6 VALIDAÇÃO E RESULTADOS

Uma questão a ser levantada ao analisar as bibliotecas SIP selecionadas e as soluções implementadas a partir delas é se estas bibliotecas são de fato comparáveis. Como descrito anteriormente, as linguagens de programação utilizadas não são as mesmas e as implementações seguem lógicas diferentes a fim de alcançar o objetivo. No entanto, o objetivo final da implementação é o mesmo (criar, negociar e estabilizar uma sessão SIP), esse fato pode ser considerado como uma análise adicional, referente a disponibilidade e facilidade de se utilizar as bibliotecas selecionadas para estudo neste trabalho. Portanto, a comparação foi realizada por meio da análise dos recursos oferecidos por cada grupo que desenvolvem as bibliotecas, e a implementação efetuada visa medir o esforço necessário para utilização das mesmas, independentemente da linguagem utilizada para construção da aplicação.

Durante este capítulo serão apresentados os resultados obtidos a partir da metodologia descrita no capítulo 4. O estudo foi realizado levando-se em conta o projeto das bibliotecas analisadas, os recursos disponíveis para auxiliar seus usuários e a solução construída.

6.1 Análise Qualitativa

A análise qualitativa foi realizada através do estudo do código fonte, da documentação e da página principal disponibilizada pelas bibliotecas. Além disso, a construção do aplicativo também contribuiu para enfatizar os resultados. Por ser uma análise subjetiva e algumas vezes até pessoal, o estudo pode gerar resultados diferentes dependendo de quem o produz, mas mesmo assim, no contexto aplicado, resulta em dados significativos quando a qualidade dos softwares analisados.

6.1.1 Usabilidade

De acordo com a metodologia especificada no capítulo 4, a análise de usabilidade visa identificar e tentar quantificar qual a facilidade com que um desenvolvedor que utilize a biblioteca descrita tem para desenvolver uma aplicação. Para isso foi descrito uma série de critérios (perguntas) para analisar se as bibliotecas estudadas atendem ou não os requisitos propostos. Os critérios utilizados para a análise de usabilidade foram divididos em quatro grupos de questões para uma melhor identificação dos tópicos abordados durante a análise. As subseções a seguir descrever os resultados obtidos para este critério.

6.1.1.1 Instalação

Em relação a instalação, o projeto PJSIP disponibiliza o seu código fonte através de sua página na internet. Além disso, é possível obter os mesmos arquivos através de um

repositório SVN[1] disponibilizado na mesma sessão de *download*, mas para isso necessário a instalação de um software de versionamento. Existem duas séries de códigos:

- série 1.0.x: possui apenas correções de bugs da versão 1.0.1. Esta versão é indicada apenas para aplicações baseadas nesta versão, pois não implementa novas funcionalidades;
- série 1.x: possui correção de bugs, novas funcionalidades e melhorias do software PJSIP. Esta versão é recomendada pelos desenvolvedores para quem desejar iniciar o desenvolvimento utilizando PJSIP.

A versão 1.7 da biblioteca PJSIP foi utilizada para o estudo neste trabalho, embora o projeto já esteja na versão 1.8.5 disponibilizada em 21 de outubro de 2010. A PJSIP não apresenta programa de instalação, o que obriga o desenvolvedor a baixar o código fonte e compilá-lo de acordo com a plataforma em que deseja utilizar. A compilação é simples e leva pouco tempo para ser realizada, além disso, não é preciso a instalação de nenhum outro software ou biblioteca adicional para que se possa utilizar a PJSIP.

A biblioteca Sofia-SIP também disponibilizam seus códigos através de sua página na internet e como alternativa possibilita o acesso ao repositório de arquivos através do software Darcs que é semelhante ao SVN. Para a versão 1.12.10 analisada, é necessária a instalação de um interpretador AWN[1] utilizado para preparar os arquivos antes de serem compilados. A compilação da biblioteca Sofia-SIP é bastante simples e rápida, levando muito pouco tempo para estar disponível para uso.

Diferentemente das outras bibliotecas, ReSIProcate não disponibiliza seus códigos através de um *(*link) em sua página principal. É necessário que o usuário instale um cliente SVN em sua máquina e a partir dai acesse o repositório com o código da biblioteca. A versão 1.6 do projeto ReSIProcate foi analisada neste estudo e não apresenta programa de instalação. Além disso, para a compilação geral do código na plataforma Windows é necessário a instalação de pacotes gráficos Direct X utilizado em alguns recursos gráficos de visualização presentes no projeto. Caso este pacote não esteja instalado ocorrerão erros de compilação em uma série de arquivos e isso impossibilitará a utilização da biblioteca. O tempo necessário para instalação e compilação do projeto é relativamente maior em relação às outras bibliotecas analisadas visto que requer alguns softwares adicionais para se alcançar o objetivo.

6.1.1.2 Exemplo de uso

A biblioteca PJSIP apresenta uma grande variedade de exemplos de uso para os seus recursos. Todos os exemplos estão presentes no projeto da pilha e são facilmente acessíveis. Além disso, apresentam nomes intuitivos para representar a sua funcionalidade e sua utilização disponibiliza comandos de ajuda para acessar todos os recursos disponíveis. Na análise realiza, o número de programas de exemplo se mostrou satisfatório para compreensão dos recursos disponíveis na biblioteca PJSIP.

Em relação aos programas de exemplos, a biblioteca ReSIProcate disponibiliza três destes em seu projeto. Estes exemplos são bem simples e abordam as funcionalidades da biblioteca de maneira bem superficial. O objetivo dos exemplos é basicamente proporcionar ao programador uma demonstração de como é possível realizar e receber chamadas, trocar mensagens de texto e, se necessário, efetuar um registro em um servidor

[1] SVN: Subversion (também conhecido por svn) é um sistema de controle de versão.

[1] AWN: é uma linguagem de programação para processamento de dados baseados em texto, sejam eles provenientes de arquivos ou de streams de dados.

SIP utilizando a biblioteca ReSIProcate. Os exemplos permitem compreender como a pilha ReSiprocate funciona, mas um maior número destes se mostra necessário para uma demonstração mais completa das funcionalidades disponíveis na biblioteca.

Na versão analisada para a biblioteca Sofia-SIP não foram encontrados programas de exemplos em seu projeto. Pesquisando sobre este fato em (CENTER, 2005), é indicado o software *SofSipCli* como referência para uso da biblioteca. Este software é um cliente VoIP/IM repleto de funcionalidades, mas está disponível apenas para a plataforma GNU/-LINUX. Na versão anterior à analisada, foi encontrado um programa de exemplo para a utilização da interface NUA que é parte do projeto Sofia-SIP Library. Este exemplo demonstra algumas das funcionalidades básicas da biblioteca, mas é de difícil acesso tendo em vista que é necessário baixar o código do projeto em uma versão anterior. Os exemplos analisados permitem compreender em parte a funcionalidade da pilha Sofia-SIP, mas para que funcionem é necessário ao programador certo grau de experiência com a manipulação desta biblioteca. Um maior número de exemplos se faz necessário para que um usuário iniciante possa ter mais clareza ao utilizar Sofia-SIP.

6.1.1.3 Programas teste

A presença de programas de teste é importante, pois a partir deles é possível detectar possíveis erros ou bugs na implementação de um software. A biblioteca PJSIP apresenta arquivos de teste para praticamente todas as funcionalidades citadas em (PJSIP.ORG, 2005). Os testes são escritos de forma simples e são suficientes para identificar o que está sendo testado em questão.

Em relação aos programas de teste, a biblioteca ReSIProcate apresenta apenas um arquivo para testar as suas funcionalidades. Este arquivo é bem simples e basicamente testa a eficiência com que a biblioteca realiza suas funções. Contudo, o projeto da pilha ReSIProcate não apresenta testes que demonstrem o correto funcionamento do tudo o que implementa, dificultando assim a detecção de possível erros ou bug em sua implementação.

Já o projeto da biblioteca Sofia-SIP apresenta uma série de programas para testar seu funcionamento. Basicamente todos os módulos da biblioteca apresentam testes detalhados e completos.

6.1.1.4 API de programação

A utilização das interfaces de programação disponibilizada pelas bibliotecas analisadas seguem uma linha parecida de passos a serem seguidos com o intuito de construir um *User Agent*. Estes passos basicamente se resumem em inicializar módulos de cada biblioteca, alocar recursos (memória) necessários e configurar o aplicativo de maneira adequada.

Para realização dos passos descritos anteriormente, a biblioteca PJSIP apresenta uma API com funções de nomes intuitivos. As funções e tipos definidos pela biblioteca utilizam prefixos que auxiliam o programador a identificar exatamente o que cada um deles representa. Mesmo assim, em relação às outras bibliotecas analisadas, PJSIP necessita de um número maior de comando para se escrever um código que realize um mesmo objetivo.

Para utilizar a pilha ReSIProcate deve-se instanciar alguns objetos que derivam de classes já definidas nos cabeçários do projeto. Estes objetos apresentam métodos com nomes muito intuitivos e auto-explicativos no que se diz respeito às suas funcionalidades, facilitando assim a sua utilização para criação de uma aplicação.

Em relação às outras bibliotecas estudadas, a Sofia-SIP apresenta funções e tipos que podem confundir o programador quanto a funcionalidade que cada comando implementa. É necessário a escrita de um número grande de funções auxiliares não implementadas pela biblioteca para se ter acesso e manipular alguns dados necessários quando se escreve uma aplicação. Em parte isso é ruim, pois dificulta a legibilidade do código criado, mas por outro lado apresenta um ponto positivo pois a API é flexível com a manipulação dos dados e possibilita ao programador uma maior customização quanto a maneira que os dados podem ser utilizados.

Um fator interessante a ser analisado quando se desenvolve um aplicativo utilizando uma biblioteca desenvolvida por terceiros é a presença de mecanismos que auxiliem a depuração do funcionamento do software criado. A possibilidade de se criar um arquivo de log que armazene eventos e informações implicitos a API da biblioteca é útil, pois disponibiliza uma maneira de depurar erros ou bugs durante a execução de um aplicativo. As bibliotecas PJSIP e Resiprocate apresentam a possibilidade de criação de estruturas para monitorar eventos internos de suas funcionalidades, já a biblioteca Sofia-SIP deixa a desejar neste quesito, deixando a cargo do programador a criação de estruturas deste tipo.

6.1.2 Documentação

De acordo com a metodologia especificada no capítulo 4, a análise de documentação visa qualificar como o conjunto de manuais gerais e técnicos explicam o funcionamento do software com um todo ou cada parte dele. Para isso foi descrito uma série de critérios (perguntas) para analisar se a biblioteca estudada atende ou não os requisitos propostos.

As três bibliotecas analisadas apresentam um manual de instalação. Este manual é importante para instruir o usuário de maneira correta e, assim, ao final dos passos descritos, a biblioteca esteja pronta para ser compilada e utilizada. Além disso, todas as pilhas utilizam o sistema doxygen (referenciar) em seus códigos fonte a fim de disponibilizar uma descrição online de suas funções, estruturas e classes.

A documentação geral da biblioteca PJSIP é bastante completa e suficiente para que a pilha seja utilizada de maneira adequada. Praticamente todas as funções disponíveis em sua API apresentam comentários e descrições. Todo o código segue o mesmo padrão de documentação, isso é muito relevante pois não confunde o desenvolvedor com um descrições diferentes presentes no projeto. Além disso, a PJSIP apresenta um tutorial online para auxiliar a familiarização do programador com o software. Tanto a descrição dos exemplos quanto a descrição do tutorial é bastante extensa e explica passo-a-passo o seu funcionamento.

Em (RESIPROCATE.ORG, 2006) é disponibilizado o código fonte documentado e os diagramas das classes presentes na API. Apesar disso, o código presente no projeto da pilha ReSIProcate quase não apresenta comentários, fazendo com que seja necessário o acesso online da documentação para esclarecimento de dúvidas referentes a utilização da biblioteca.

O código dos exemplos presentes na biblioteca ReSIProcate são pouco comentados, o que resulta em uma documentação insuficiente para o entendimento total de sua funcionalidade. Durante o estudo não foi encontrado nenhum tutorial para a utilização da pilha. De modo geral a documentação da biblioteca é pouco detalhada podendo trazer dificuldade a usuários com pouca experiência com esta pilha.

A biblioteca Sofia-SIP apresenta seu código fonte bem documentado. O usuário pode acessar a documentação tanto nos arquivos do projeto quanto na página principal da pilha. Em relação às outras bibliotecas analisadas, a Sofia-SIP é a que apresenta um tutorial de

maior qualidade. Apesar disso, os exemplos disponíveis para sua utilização não apresentam comentários, o que praticamente obriga o programador a utilizar o tutorial online da biblioteca. Mesmo assim, a Sofia-SIP pode ter sua documentação qualificada com suficientemente boa, devido ao seu tutorial, mesmo sendo necessário um tempo maior para se construir uma aplicação.

6.1.3 Suporte

Em relação ao suporte prestado pelos grupos que desenvolvem as bibliotecas analisadas, pode-se dizer que apresentam avaliações parecidas. Nenhuma das bibliotecas analisadas apresenta um fórum online para acesso a informações e dúvidas de desenvolvedores em geral, o que prejudica em parte o acesso rápido a questões técnicas referentes ao desenvolvimento de aplicações.

Os grupos pjsip.org (desenvolvedor da biblioteca PJSIP), resirocate.org (desenvolvedor da biblioteca ReSIProcate) e Nokia Reserch Center (desenvolvedor da biblioteca Sofia-SIP) disponibilizam através de seus sites a possibilidade de um cadastro a lista de emails para discussão. Através desta lista é possível postar dúvidas, receber informações e publicações referentes à utilização e desenvolvimento das bibliotecas. Apesar de a lista de email ser similar a um fórum, esta não apresenta uma forma simples de acesso a tópicos específicos, obrigando o usuário a navegar através de mensagens postadas para encontrar as informações necessárias.

A página principal da biblioteca PJSIP apresenta uma lista de perguntas freqüentes (FAQ) bastante completa, contendo mais de cem perguntas e respostas. Já a biblioteca ReSIProcate apresenta uma lista de FAQ mais modesta, com apenas onze questões de dúvidas comuns. Uma lista mais completa reduz o número de perguntas semelhantes repetidamente colocadas, facilitando assim o acesso de questões mais específicas. Por fim, a biblioteca Sofia-SIP não apresenta FAQ em sua página principal, obrigando seus usuários a procurar respostas a perguntas de comum interesse na lista de e-mails.

As páginas de internet das bibliotecas analisas se mostraram de fácil acesso e bastante objetivas. Através delas é possível ter alcance a todas as informações referentes a cada projeto, não sendo necessários acessos a páginas de terceiros para mais dados.

6.1.4 Resultados obtidos na avaliação qualitativa

De acordo com os critérios utilizados para avaliar a qualidade das bibliotecas estudas neste trabalho, foram geradas tabelas que visam responder os quesitos adotados para a comparação. As tabelas contidas no Apêndice B contêm a pontuação obtida por cada biblioteca segundo os critérios de usabilidade, documentação e suporte. Através desta pontuação, foi construído o gráfico, demonstrado na figura 6.1, que contém o percentual de qualidade obtido segundo a metodologia utilizada neste estudo comparativo.

Como demonstrado na figura 6.1, a pilha PJSIP obteve uma maior pontuação em relação às outras bibliotecas analisadas. A documentação da biblioteca PJSIP se mostrou bastante completa, e este critério pode ser citado com um ponto forte quando se deseja escolher uma biblioteca para uma implementação baseada no protocolo SIP. Além disso, a pilha obteve um bom desempenho nos quesitos de usabilidade, que dizem respeito principalmente aos recursos oferecidos em relação aos programas de teste, exemplos e a API disponíveis para auxiliar desenvolvedores que desejam utilizar PJSIP.

A biblioteca ReSIProcate obteve um bom desempenho em relação à sua usabilidade. A pilha se mostrou bastante simples em relação a facilidade oferecida para construção de aplicações. Mesmo assim, deixa a desejar quanto ao quesito de documentação, onde se

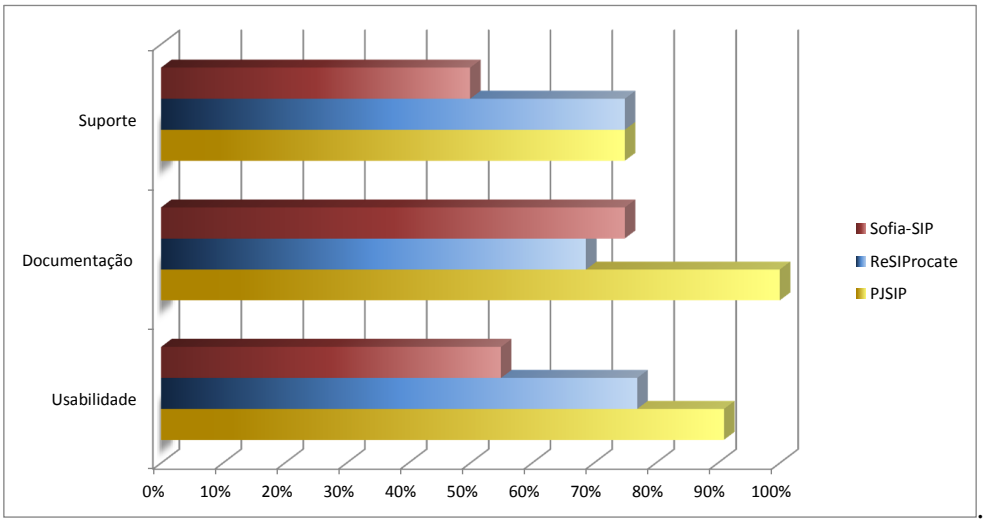

Figura 6.1: Gráfico comparativo da análise qualitativa

mostrou pobre em relação às demais bibliotecas. O código fonte do projeto analisado não apresenta muitos comentários referentes às classes e métodos oferecidos pela API, obrigando seus usuários a acessarem a documentação online para consultar possíveis dúvidas. Além disso, a documentação presente na página da biblioteca é um pouco confusa e exige certo grau de familiaridade com o código fonte para ser efetiva.

Por último, a pilha Sofia-SIP obteve um pontuação inferior às demais bibliotecas analisadas neste trabalho. Apesar de apresentar uma documentação satisfatória, sua usabilidade deixou a desejar. A maior dificuldade encontrada para sua utilização foi o número reduzido de exemplos para uso da API, obrigando um usuário pouco familiarizado com esta pilha a fazer uso de seu tutorial para auxiliar o desenvolvimento de seus aplicativos.

Em relação ao critério de suporte analisado neste trabalho, as três bilbliotecas apresentaram uma avaliação parecida. A pilha Sofia-SIP obteve uma pontuação inferior às demais pelo fato de não apresentar uma lista de perguntas freqüentes em seu *website*. As informações disponíveis nas páginas de cada biblioteca são parecidas e o suporte oferecido pelos grupos que desenvolvem as pilhas é bastante similar.

6.2 Análise Quantitativa

Passando para a análise quantitativa, é esperado encontrar na prática os conceitos descritos na análise qualitativa realizada anteriormente. Essa análise é feita sobre os códigos fonte e a execução do aplicativo desenvolvido, visando demonstrar as características da utilização das bibliotecas no contexto deste estudo.

A análise de código escrito visa identificar a complexidade e o esforço necessários para a construção de módulos de um aplicativo com funcionalidades similares. Mesmo que os módulos sejam implementados através de linguagens diferentes (no caso deste trabalho C e C++), diferenças expressivas no número de linhas de código gerado podem estimar o esforço e complexidade de cada implementação.

Como descrito no capítulo 4, serão utilizadas as métricas de LOC (*Lines of Code*), eLOC (*Effective Lines of Code*) e lLOC (*Logical Lines of Code*) para metir a quantidade de linhas de código geradas a partir de cada implementação. Para analisar o código, foi

utilizada a ferramenta RSMWizard[1], capaz de contabilizar as informações de um projeto em ambas as linguagens. O gráfico ilustrado através da figura 6.2 demonstra os resultados obtidos para esta comparação.

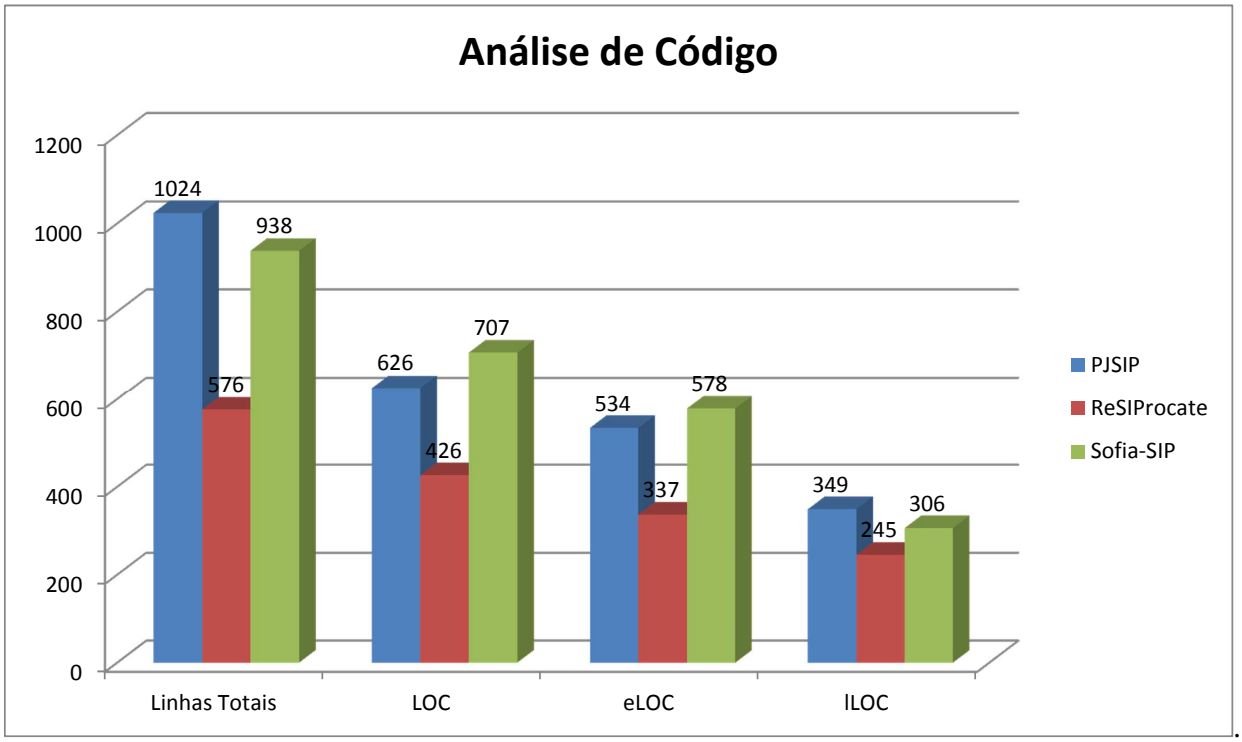

Figura 6.2: Gráfico contendo a análise de código

De acordo com a figura 6.2, os códigos escritos através do uso das bibliotecas PJSIP e Sofia-SIP apresentam um número total de linhas de código quase duas vezes maior do que o código escrito utilizando a biblioteca ReSIProcate. Esta diferença pode ser explicada pela necessidade de se utilizar arquivos de cabeçalho (*headers*) para declaração de estruturas e funções nas implementações utilizando a linguagem C, fazendo com algumas partes do código sejam replicadas. Para a implementação utilizando a linguagem C++ as definições de classes e métodos foram escritas no mesmo arquivo, fazendo com que o número final de linhas de código fosse relativamente menor do que as outras implementações realizadas. Outro fato relevante a ser mencionado é que o número total de linhas de código leva em consideração todo o código escrito, incluindo comentários, linhas em branco, identações diferentes, etc.

A medida de LOC (*Line of Code*) obtida através do software RSMWizard, elimina os comentários e linhas em branco presentes no arquivo de código. Embora seu resultado seja muitas vezes arbitrário em virtude de não representar em instruções realmente computáveis, esta medida é importante para expressar o esforço necessário para se escrever uma aplicação. De acordo com os resultados obtidos, as implementações utilizando as bibliotecas PJSIP e Sofia-SIP geraram um código cerca de vinte porcento maior do que a implementação utilizando ReSIProcate.

As medidas de eLOC (*Effective Lines of Code*) e lLOC (*Logical Lines of Code*) são mas efetivas em relação à expressividade do código realmente computável. De acordo

[1]RSMWizard: disponível em <http://msquaredtechnologies.com/>. Acesso em novembro de 2010.

com os resultados obtidos, a implementação utilizando Sofia-SIP necessitou de um número maior de linhas de código do que as demais para a medida de eLOC. Embora este resultado apresente pouca diferença em relação à implementação utilizando PJSIP, a diferença em relação a ReSIProcate ainda se mostrou relavante. A medida de lLOC pode ser considerada a mais efetiva em relação ao número de comandos necessários para a construção do aplicativo. Os resultados demonstram que a solução com maior número de comandos foi escrita com a biblioteca PJSIP. Este resultado pode ser explicado pelo fato que a API disponível nesta biblioteca requer a utilização de muitas funções implementadas pela pilha, resultando em pouca flexibilidade ao programador para escrever soluções mais customizadas. Já a biblioteca Sofia-SIP disponibiliza mais flexibilidade ao programador, necessitando de menos chamadas de funções presentes em sua API, deixando a cargo do usuário escolher a melhor forma para sua utilização. Por último a biblioteca ReSIProcate se mostrou mais fácil em relação a sua utilização, necessitando de menos esforço para gerar resultados semelhante. Além de ser flexível, a complexidade de sua utilização é abstraída através de sua API, facilitando seu uso e deixando a cargo do desenvolvedor a tarefa de implementar apenas os recursos que realmente interessam para sua aplicação.

7 CONCLUSÃO

Atualmente, o protocolo SIP vem sendo largamente utilizado em aplicativo voltados a telefonia IP (VoIP). Devido a sua simplicidade em relação aos outros protocolos de sinalização, este trabalho visou comparar algumas das principais bibliotecas de funções que implementam o protocolo SIP a fim de realizar uma adaptação de algumas funcionalidades presente neste protocolo a um software de videoconferência em desenvolvimento.

Durante o estudo comparativo apresentado neste trabalho foi visto que existem um série de bibliotecas que implementam o protocolo SIP. Como não há espaço para uma comparação ampla entre todas estas, foram selecionadas algumas bibliotecas que se adequassem à implementação prática proposta. As bibliotecas PJSIP, ReSIProcate e Sofia-SIP foram selecionadas para este estudo.

A comparação das bibliotecas se deu através do uso de métricas qualitativas e quantitativas apresentadas no capítulo de metodologia. Com isso se tentou avaliar quais as facilidades e dificuldades que um programador irá encontrar quando desejar construir uma aplicação que faça uso de uma dessas bibliotecas.

Qualitativamente foi visto que a biblioteca PJSIP apresentou um melhor resultado em relação aos critérios adotados. Por apresentar uma documentação bastante completa tanto em seus códigos quanto em sua página *online*, obteve uma nota muito boa nos quesitos propostos para este critério. Outro fator determinante para sua qualificação foi a presença de vários exemplos de uso e testes de uso em seu projeto, o que facilita em muito a criação de aplicativos quando não se tem um domínio pleno da API utilizada.

A biblioteca ReSIProcate apresentou uma API mais amigável em relação as demais bibliotecas. Um fator negativo encontrado nesta em biblioteca foi sua documentação não muito completa. Este fator foi determinante para que sua análise qualitativa não apresentasse resultados tão bons quanto a PJSIP. Outro fato que deixou a desejar foi a presença diminuta de testes de uso em seu projeto, o que pode acarretar em problemas quando se deseja testar funcionalidades da API.

Já a biblioteca Sofia-SIP obteve resultados inferiores às demais bibliotecas em relação à análise qualitativa. Apesar de apresentar uma documentação bastante razoável, o fator determinante para seu menor desempenho foi o baixo número de exemplos de uso que seu projeto disponibiliza. Isso dificulta bastante sua utilização, pois requer uma maior familiaridade do programador em relação à API.

Em relação às métricas de códigos adotas durante a análise quantitativa foi possível perceber que em relação ao esforço necessário para se criar uma aplicação, a biblioteca ReSIProcate obteve um resultado melhor. Apesar de ser escrita em uma linguagem diferente em relação às demais, o número de comandos necessários para escrita de um código funcional apresentou um número inferior às outras bibliotecas o que indica uma menor probabilidade de ocorrerem erros durante esta tarefa.

Por sua vez, as bibliotecas PJSIP e Sofia-SIP apresentaram um resultado semelhante em relação ao número de linhas de código e comandos necessários para a geração de um aplicativo com funcionalidades parecidas. Apesar disto, o que se pode afirmar em relação às duas implementações efetuadas é que o modo com que as APIs são utilizadas seguem lógicas bastante distintas. A biblioteca PJSIP oferece pouca flexibilidade ao programador que a utiliza, fazendo com que utilize grande parte dos recursos oferecidos por sua API a fim de manipular os dados do programa. Este fato pode estar relacionado ao número elevado de plataformas que a biblioteca suporta, necessitando um tratamento mais conciso de seus dados a fim de apresentar a portabilidade prometida. Já a biblioteca Sofia-SIP disponibiliza uma maior flexibilidade em relação à utilização de seus recursos. Para sua utilização o programador faz uso de um número menor de funções provenientes desta API, mas em contra partida é necessário que o programador crie mecanismos próprios para auxiliar o acesso aos dados produzidos pela biblioteca.

Portanto, através deste trabalho foi possível identificar alguns fatores que podem ser importantes em relação à escolha de uma biblioteca que implemente as funcionalidades prevista no protocolo SIP. Embora não mencionado anteriormente, as bibliotecas estudadas apresentam propostas diferentes em relação a sua utilização. A biblioteca PJSIP apresenta uma solução mais completa em relação à criação de softwares que desejam utilizar seus recursos, principalmente em relação à criação de *softfones*. Já as bibliotecas ReSIProcate e Sofia-SIP visam disponibilizar ferramentas para criação de entidades SIP tomando como base apenas a sinalização, não dando suporte a transporte de mídia e codecs necessários.

Por fim, como trabalhos futuros, são sugeridos uma análise mais detalhada em relação às funcionalidades disponibilizadas em cada biblioteca. Esta análise pode ter como objetivo não apenas a criação de *User Agents*, mas também a criação de *Proxy Servers* e outras entidades mencionadas em (ROSENBERG et al., 2002). Outra sugestão é a comparação de bibliotecas SIP que dêm suporte a construção de aplicações para dispositivos móveis, possibilitando assim sessões de mídia e IM para estes dispositivos.

REFERÊNCIAS

BACLOR, J. E. **OpenSIPStack.org Home Page**. Disponível em: <http://www.opensipstack.org>. Acesso em Novembro de 2010.

BASILI, V. R.; CALDIERA, G.; ROMBACH, D. **The Goal Question Metric Approach. Encyclopedia of Software Engineering. Volume 1**. 2.ed. [S.l.]: John Wiley and Sons, 2002.

CAMARILLO, G. **SIP Demystified**. 1.ed. [S.l.]: McGraw-Hill Professional, 2001. 320p.

CENTER, N. R. **Sofia-Sip Library Home Page**. Disponível em: <http://sofia-sip.sourceforge.net/>. Acesso em Novembro de 2010.

FARIAS, C. M. de; LUSTOSA, L. C. G.; A. RODRIGUES, P. H. de; MOREIRA, P. R. **Combinação de pilhas de protocolo para a construção de um softphone SIP**. [S.l.]: WebMedia 2008 Proceedings of the 14th Brazilian Symposium on Multimedia and the Web, 2008.

FENTON, N. E.; PFLEEGER, S. L. **Software Metrics**: a rigorous and practical approach. 2.ed. Londres: Course Technology, 1998. 656p.

HANDLEY, M.; UCL; JACOBSON, V.; DESIGN, P.; PERKINS, C.; GLASGOW, U. of. **SDP**: session description protocol. [S.l.]: IETF, 2006. n.4566. (Request for Comments).

ISO/IEC. **ISO/IEC 9126. Software engineering – Product quality**. [S.l.]: ISO/IEC, 2001.

LEOPOLDINO, G. M.; MEDEIROS, R. C. M. de. **H.323**: um padrão para sistemas de comunicação multimídia baseado em pacotes. Acesso em Novembro de 2010.

MINISIP.ORG. **Minisip**. Disponível em: <http://www.minisip.org>. Acesso em Novembro de 2010.

MOIZARD, A. **oSIP Library**. Disponível em: <http://www.gnu.org/software/osip/>. Acesso em Novembro de 2010.

MOIZARD, A. **eXosip2 - the eXtended osip Library**. Disponível em: <http://savannah.gnu.org/projects/exosip>. Acesso em Novembro de 2010.

PJSIP.ORG. **PJSIP - Open Source SIP Stack Home Page**. Disponível em: <http://www.pjsip.org/>. Acesso em Novembro de 2010.

RESIPROCATE.ORG. **reSIProcate projects Home Page**. Disponível em: <http://www.resiprocate.org/>. Acesso em Novembro de 2010.

ROSENBERG, J.; SCHULZRINNE, H.; CAMARILLO, G.; JOHNSTON, A.; PETERSON, J.; SPARKS, R.; HANDLEY, M.; SCHOOLER, E. **SIP**: session initiation protocol. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, RFC 3261 (Proposed Standard).

SANT'ANNA, C. N. **Manutenibilidade e Reusabilidade de Software Orientado a Aspectos**: um framework de avaliação. 2004. Dissertação (Mestrado em Ciência da Computação) — Pontifícia Universidade Católica do Rio de Janeiro.

SCHULZRINNE, H.; UNIVERSITY, C.; CASNER, S.; DESIGN, P.; FREDERICK, R.; INC., B. C. S.; JACOBSON, V. **RTP**: a transport protocol for real-time applications. [S.l.]: IETF, 2003. n.3550. (Request for Comments).

STORCH, M. M. **Proposta de avaliacão da qualidade de produtos de software utilizando a norma ISO/IEC 9126**. 2000.

TANENBAUM, A. S. **Computer Networks**. 4.ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2003.

WALLINGFORD, T. **Switching to VoIP**. 1.ed. Gravenstein Highway North, Sebastopol, CA: O'Reilly Media, Inc., 2005. 477p.

ANEXO A LISTA COMPLETA DE RESPOSTAS SIP

100	Trying	413	Request entity too large
180	Ringing	414	Request-URI too large
181	Call is being forwarded	415	Unsupported media type
182	Queued	420	Bad extension
183	Session progress	480	Temporarily not available
200	OK	481	Call leg/transaction does not exist
202	Accepted	482	Loop detected
300	Multiple choices	483	Too many hops
301	Moved permanently	484	Address incomplete
302	Moved temporarily	485	Ambiguous
305	Use proxy	486	Busy here
380	Alternative service	487	Request cancelled
400	Bad request	488	Not acceptable here
401	Unauthorized	500	Internal servere error
402	Payment required	501	Not implemented
403	Forbidden	502	Bad gateway
404	Not found	503	Service unavailable
405	Method not allowed	504	Gateway time-out
406	Not acceptable	505	SIP version not supported
407	Proxy authentication required	600	Busy everywhere
408	Request time-out	603	Decline
409	Conflict	604	Does not exist anywhere
410	Gone	606	Not acceptable
411	Length required		

Figura A.1: Tabela completa de respostas SIP

APÊNDICE A TABELAS APÓS ANÁLISE QUALITATIVA

AVALIAÇÃO DE USABILIDADE			
PERGUNTAS	PJSIP	ReSIProcate	Sofia-SIP
1. A biblioteca possui programa de instalação?	0	0	0
2. O tempo necessário para sua instalação é satisfatório?	2	2	2
3. Para a utilização da biblioteca não é necessária a instalação de softwares/bibliotecas adicionais além da IDE de programação?	2	1	1
4. A biblioteca possui programas de exemplo?	2	2	1
5. Os exemplos permitem compreender como a biblioteca funciona?	2	2	1
6. Os exemplos de uso são facilmente acessíveis?	2	2	1
7. O número de programas exemplo é satisfatório?	2	1	1
8. A biblioteca apresenta programas de teste?	2	2	2
9. O número de programas para teste de funcionalidades da biblioteca é satisfatório?	2	1	2
10. A API da biblioteca faz uso de nomes satisfatórios para suas funções?	2	2	1
11. A biblioteca disponibiliza a utilização de arquivos de log para facilitar a depuração de eventuais erros?	2	2	0
NOTAS RECEBIDAS:	20	17	12

Figura A.1: Resultados obtidos após avaliação de Usabilidade.

AVALIAÇÃO DE DOCUMENTAÇÃO			
PERGUNTAS	**PJSIP**	**ReSIProcate**	**Sofia-SIP**
1. A biblioteca apresenta manual de instalação?	**2**	**2**	**2**
2. A biblioteca faz uso de alguma ferramenta ou padrão para documentação?	**2**	**2**	**2**
3. A biblioteca possui um tutorial de boa qualidade?	**2**	**0**	**2**
4. Todas as funções da biblioteca apresentam documentação?	**2**	**2**	**2**
5. Os arquivos de código fonte seguem o mesmo padrão de documentação?	**2**	**2**	**2**
6. Os exemplos de uso da biblioteca apresentam documentação?	**2**	**1**	**0**
7. A documentação dos exemplos de uso é suficiente para o entendimento dos mesmos?	**2**	**1**	**0**
8. A documentação geral da biblioteca é suficiente para que o desenvolvedor possa utilizá-la de maneira adequada?	**2**	**1**	**2**
NOTAS RECEBIDAS:	16	11	12

Figura A.2: Resultados obtidos após avaliação de Documentação.

AVALIAÇÃO DE SUPORTE			
PERGUNTAS	**PJSIP**	**ReSIProcate**	**Sofia-SIP**
1. O grupo que desenvolve a biblioteca disponibiliza um fórum de discussão online?	**0**	**0**	**0**
2. O grupo que desenvolve a biblioteca disponibiliza uma lista de emails para publicação de informações referentes a seu desenvolvimento ou discussões sobre o projeto?	**2**	**2**	**2**
3. A página principal do projeto da biblioteca apresenta lista de FAQ?	**2**	**2**	**0**
4. A página principal do projeto da biblioteca é objetiva e de fácil acesso?	**2**	**2**	**2**
NOTAS RECEBIDAS:	6	6	4

Figura A.3: Resultados obtidos após avaliação de Suporte