Rafael Ossamu Togo

Implementação e avaliação de cenário de convergência telefonia-rede integrando serviços de VoIP e vídeo chamada com o uso de WebRTC

São José – SC

Agosto / 2015

Rafael Ossamu Togo

Implementação e avaliação de cenário de convergência telefonia-rede integrando serviços de VoIP e vídeo chamada com o uso de WebRTC

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. M.Sc. Arliones Stevert Hoeller Junior

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Agosto / 2015

Monografia sob o título *"Implementação e avaliação de cenário de convergência telefonia-rede integrando serviços de VoIP e vídeo chamada com o uso de WebRTC"*, defendida por Rafael Ossamu Togo e aprovada em 20 de Agosto de 2015, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. M.Sc. Arliones Stevert Hoeller Junior
Orientador - IFSC

Prof. M.Sc. Ederson Torresini
IFSC

Prof. M.Sc. Ramon Mayor Martins
IFSC

Na vida, a única coisa que pode ser conquistada sem esforço, é o fracasso.

Desconhecido

Agradecimentos

Agradeço a todos meus amigos que sempre me incentivaram a continuar, ao apoio da minha família, a minha namorada e companheira Patricia Sousa Rodrigues pela paciência, ao suporte dos professores envolvidos, um agradecimento especial ao Paulo Vitor Chirolli de Almeida e Moisés Moselle, duas pessoas que me auxiliaram em todo o andamento do trabalho e que foram essenciais para o seu término e um obrigado a Deus por me ajudar nas horas de dificuldades.

Resumo

O WebRTC *Web Real Time Communication* permite a transmissão de áudio vídeo e compartilhamento de dados entre qualquer navegador, de maneira nativa. Ou seja, não há necessidade de instalar *plugins* ou *software* de terceiros, evitando assim qualquer problema de falhas de segurança nos *plugins* utilizados, ou instalação de *softwares* mal intencionados ao instalar estes *plugin*s. O WebRTC não veio para adicionar novos serviços à rede, mas sim tornar a sua utilização prática e fácil.

Este documento aborda as tecnologias necessárias para o desenvolvimento de uma aplicação, que tem como objetivo realizar a interação de serviços de VoIP utilizando o WebRTC e Asterisk, e criar novas funcionalidades na aplicação que sejam úteis aos usuários. Será abordado também as vantagens e desvantagens frente as atuais tecnologias, e avaliar o desempenho do sistema implementado.

Palavras-chaves: WebRTC, VoIP, SIPML5, Asterisk, Node.js, Soluções Web.

Abstract

The WebRTC (Web Real Time Comunication) allows the transition of audio, video and data sharing with any browser, by native nature, it means that none plugins or third party softwares need to be installed, avoiding any security problems in the plugins, or the installation of malicious software that came with those plugins. The WebRTC did not come to add new services to the network, but make its use practical and easy.

This document discusses the technologies needed to develop an application, which aims to incorporate the interaction of VoIP services using the WebRTC and Asterisk, and create new features in the application that are useful to users. It will also be discussed the advantages and disadvantages forward current technologies, and evaluate the performance of the implemented system.

Key-words: WebRTC, VoIP, SIPML5, Asterisk, Node.js, Web Solutions.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Entendemos que a Internet revolucionou a forma como as pessoas vivem e se relacionam, porém a maneira em que hoje a enxergamos a *web*, demandou um certo tempo de amadurecimento, no que diz respeito à organização, estruturação e padronização.

Em 12 de março de 1989, Tim Berners-Lee divulgou no interior da Organização Europeia para a Pesquisa Nuclear (CERN) um projeto baseado no conceito de hipertexto para facilitar o compartilhamento e a atualização de informações apenas entre os pesquisadores da própria organização. Deu-se então o surgimento do que viria a se tornar a *World Wide Web* (WWW).

Com a *web* em processo de evolução, em 1994 surge a W3C (World Wide Web Consortium) que torna-se o órgão oficial para a gestão da *Internet* e padrões da *web*, como HTML (*HyperText Markup Language*) e CSS (*Cascading Style Sheets*), em todo o mundo, tornando a página *web* melhor estruturada e organizada.

Com o passar do tempo, diversas diretrizes, recomendações e normas da W3C foram surgindo, tornando uma página *Web* um ambiente maduro para desenvolvimento, permitindo a diversos desenvolvedores criarem suas próprias aplicações.

No ano de 2012, com a ajuda da W3C e da IETF (*The Internet Engineering Task Force*) foi possível desenvolver(ainda em fase de amadurecimento) uma tecnologia que permite comunicação em áudio e vídeo diretamente entre *web* browsers, resultando no chamado *Web Real Time Communication* (WebRTC). Esta tecnologia permite que haja comunicação em tempo real, com voz, vídeos e dados, utilizando apenas navegadores *web*, não havendo necessidade de instalação de *plugins* ou *softwares*, como é o caso do Skype, eBuddy e Nimbuzz, nem da intermediação de servidores para troca de dados.

Em uma época onde temos cada vez mais serviços rodando na nuvem, essa tecnologia tem tudo para se tornar a base de soluções de comunicação em tempo real, fazendo com que muitas empresas hoje consolidadas busquem pelo mesmo tipo de solução para não perderem espaço no mercado.

1.1 Motivação

O fato do WebRTC permitir estabelecer comunicação por som e imagem (videoconferência), de *browser* para *browser*, sem necessidade de nenhum outro *plugin* ou *software*, já o torna atraente para as futuras aplicações, pois implica em menos burocracia ao cliente final. Ou seja, a tendência para as próximas aplicações que utilizarem esta tecnologia, é tornar as comunicações entre os usuários, cada vez mais prática e eficaz. É por esse tipo de solução que os usuários tem buscado.

Este trabalho também tem como motivação, dar continuidade ao TCC desenvolvido anteriormente no IFSC-SJ por Felipe Borges intitulado *WebRTC: Estudo e Análise do Projeto* (BORGES, 2013).

1.2 Objetivos

O objetivo deste trabalho é implementar e avaliar um cenário típico de convergência entre telefonia e rede integrando serviços de VoIP e vídeo-chamadas, utilizados a partir de navegadores *Web* com suporte ao WebRTC. Neste cenário será desenvolvido uma aplicação onde um usuário encontrará outros que estarão disponíveis (*online*) em sua lista de contatos, para poder iniciar uma chamada e um sistema de *chat* em tempo real.

1.2.1 Objetivos Específicos

- Estudar a interação do sipML5 com o Asterisk e utilizá-los na solução proposta;
- Desenvolver/buscar um mecanismo de divulgação de usuários registrados (ramais declarados no arquivo de configuração do Asterisk);
- Desenvolver um sistema de *chat* entre os usuários ativos no sistema;
- Criar uma interface amigável e intuitiva aos usuários, para lidar com a aplicação desenvolvida;

1.3 Organização do Texto

Este trabalho está organizado em 6 Capítulos. O capítulo 2 aborda os conceitos básicos sobre a tecnologia WebRTC, o *software* de código aberto que implementa os recursos encontrados

em um PABX convencional, porém utilizando a tecnologia VoIP, o Asterisk, e uma explicação sobre todas as tecnologia que foram necessárias para o desenvolvimento da aplicação, incluindo o tipo de sinalização utilizada. Já no capítulo 3 é definido o sistema proposto e uma visão geral do sistema, relatando toda a metodologia utilizada. O capítulo 4 apresenta como foi implementado o sistema, juntamente com as dificuldades encontradas ao longo do desenvolvimento. No capítulo 5 são apresentados os resultados obtidos. Por fim, no Capítulo 6, é apresentado as conclusões deste trabalho.

2 *Fundamentação Teórica*

Este capítulo apresenta a fundamentação teórica do trabalho, inclui o estudo e descrição das tecnologias que serão importantes para o desenvolvimento do projeto.

2.1 WebRTC

O WebRTC é um projeto de código aberto que permite a comunicação em tempo real, com voz, vídeos e dados, utilizando apenas navegadores *web*, não havendo necessidade de instalação de *plugins* ou *softwares*. (BORGES, 2013)

Oferece aos desenvolvedores meios para desenvolver aplicações em tempo real de maneira prática e auxilia na construção de uma plataforma RTC (*Real-Time Communications*) que funciona em diversos navegadores através de múltiplas plataformas.

O papel fundamental do WebRTC é disponibilizar funções e recursos para que o desenvolvedor *web* possa implementar uma aplicação que funcione como um telefone ou uma unidade de vídeoconferência, pois o WebRTC nada mais é que um *media engine*[1] que faz interações com API's Javascript e com o próprio sistema operacional do usuário, para ter acesso aos recursos de câmera e microfone.

Outras funções que são de responsabilidade de um navegador com suporte a WebRTC é a codificação/decodificação das mídias que serão utilizadas, bem como funções de processamento de mídia como cancelamento de eco entre outras.

A arquitetura do WebRTC é constituida por duas camadas. A primeira é denominada *Web* API, onde são disponibilizada funções para que os desenvolvedores *Web* possam desenvolver suas aplicações(BORGES, 2013). A segunda é o WebRTC C++ API, a qual é utilizada para o desenvolvimento de browsers. A Figura 2.1 demonstra a arquitetura do WebRTC, com suas camadas e blocos.

[1] Responsável por realizar o tratamento da mídia.

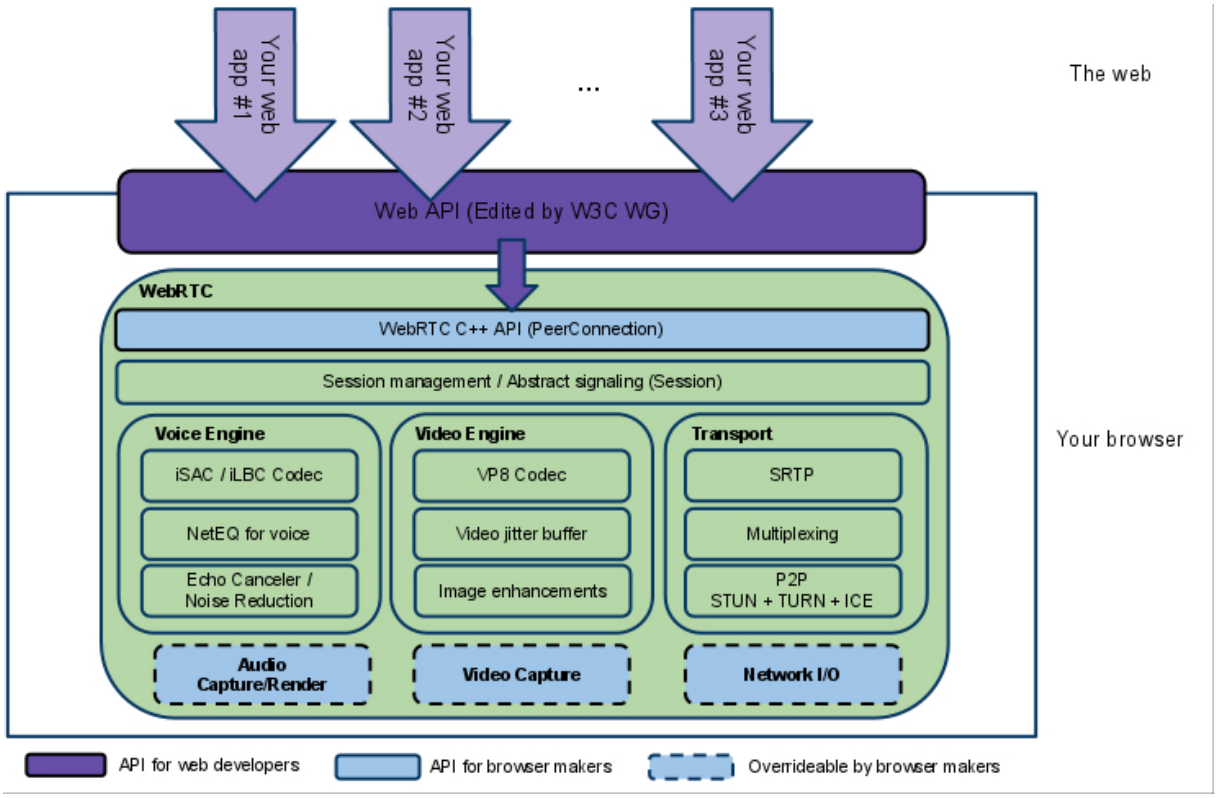

Figura 2.1: Arquitetura do WebRTC

Fonte: webrtc.org, 2012

2.1.1 Web API

Utilizando o Javascript, esta API permite o desenvolvimento de aplicações de comunicação em tempo real em *web browsers*. Podemos dividir em três partes: *MediaStream*, *PeerConnection* e *DataChannels*.

MediaStream

Utilizando a função *getUserMedia()*, esta API é responsável por solicitar acesso ao microfone e câmera do usuário para a geração de uma *stream* de dados. Esta *stream* será enviada utilizando o protocolo de transporte *Secure Real-time Transport Protocol* (SRTP), no qual é umas das exigências no uso do WebRTC. Exemplo de funcionamento na Figura 2.2. Vale reforçar, que nesta imagem deve ser subtituido o RTP pelo SRTP.

O código abaixo (Figura 2.3), mostra de maneira simples, como é feita a captura da mídia de um usuário. Neste exemplo, é solicitado apenas áudio para ser enviado para outro destino.

Na última linha, o método *navigator.getUserMedia()*, solicita o acesso ao microfone do usuário. Logo em seguida chama a função *gotStream()* para gerar uma stream de dados, que

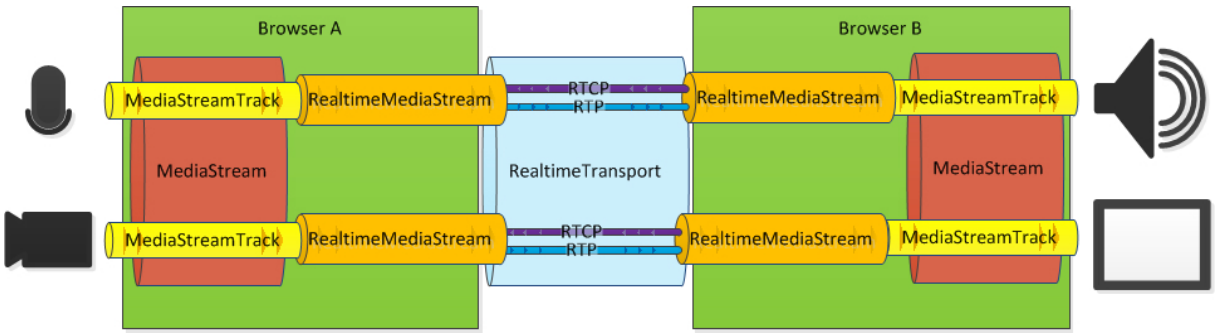

Figura 2.2: Captura e envio da stream para outro usuário.

Fonte: webrtc-experiment.com, 2014

```
function gotStream(stream) {
    window.AudioContext = window.AudioContext ||
window.webkitAudioContext;
    var audioContext = new AudioContext();

    // Create an AudioNode from the stream
    var mediaStreamSource =
audioContext.createMediaStreamSource(stream);

    // Connect it to destination to hear yourself
    // or any other node for processing!
    mediaStreamSource.connect(audioContext.destination);
}

navigator.getUserMedia({audio:true}, gotStream);
```

Figura 2.3: Função de captura de mídia.

Fonte: html5rocks.com, 2012

pode ser enviado a algum destino específico.

PeerConnection

Esta API é quem estabelece uma conexão entre dois browsers, faz o controle do codec que será utilizado, criptografia e gerenciamento de banda. Ou seja, protege os desenvolvedores *web* de diversas complexidades que existem, ao se fazer uma conexão *browser-to-browser* Para estabelecer esta conexão, é necessário um canal para a sinalização. Este é implementando num servidor, utilizando WebSockets ou XML-HttpRequest. (BORGES, 2013)

DataChannel

Para a transferência de dados diretamente de um ponto a outro, utilizamos a API DataChannel. São exemplos de dados que podem ser transferidos:

- Transferência de arquivos
- Mensagens instantâneas
- Compartilhamento de tela

Com o emprego deste mecanismo tem-se como expectativa o desempenho e baixa latência de conexão entre dois clientes. Pode-se utilizar tanto TCP (*Transmission Control Protocol*), causando maior latência porém com entrega garantida, ou UDP (*User Datagram Protocol*), com menor latência mas com possível perda de pacotes.

2.1.2 WebRTC C++ API

Esta API é voltada exclusivamente para o desenvolvimento de browsers, na implementação do WebRTC. A WebAPI é *open source* e está disponível para qualquer desenvolvedor que queira realizar a integração de seus produtos com os padrões WebRTC. Este trabalho não trabalhará em cima desta API.

2.1.3 *Voice Engine*

Como foi visto na imagem 2.1, o Voice Engine é um dos mecanismos que formam o WebRTC. É responsável desde a captura do áudio da placa de som, tratamento e envio para a interface de rede. Cancelamento de eco, redução de ruído e uso de codecs específicos. (BORGES, 2013)

2.1.4 *Video Engine*

Responsável pela captura da imagem da *WebCam*, para que esta seja enviada pela *web*. Também é responsável pelo seu tratamento, reduzindo a quantidade de ruído da imagem. Até hoje não foi definido qual codec de vídeo que será utilizado, pois ainda existe uma grande disputa comercial em torno deste assunto. A disputa hoje está entre o VP8 e o H.264.

Em questão de qualidade e consumo de bateria, o H.264 é considerado superior, pois os processadores de vídeo atuais possuem *hardware* dedicados para codificá-los, ao contrário do

VP8, onde a codificação é feita através de *software*, consumindo uma quantidade considerável da bateria. Em favor do VP8 e uma das premissas do WebRTC desde o seu princípio, é que este está livre de *royalties*(BORGES, 2013).

2.1.5 *Transport*

Em relação ao transporte da mídia, o WebRTC faz uso do SRTP (que será explicado na subseção 2.4.1), que tem como função criptografar a mídia para que esta não seja decodificada por terceiros.

Para clientes que estão por trás de NAT (*Network address translation*), há necessidade da utilização dos protocolos ICE, STUN e/ou TURN para auxiliar no envio da mídia.

2.2 *Network Address Translation (NAT)*

NAT é um mecanismo de tradução de endereços de rede, com o objetivo de possibilitar conectividade entre nós de uma rede privada com a internet.

À medida com que a rede de internet foi crescendo, notou-se que o bloco de endereçamento IP poderia se esgotar em algum momento. Viu-se então a necessidade de criar um método que pudesse desacelerar esse crescimento, sem que fosse necessário alterar toda a topologia já existente, como é o caso do IPv6 [2].

Seu funcionamento se dá através de um roteador que atua como *gateway* que comuta todo o tráfego entre uma rede privada com a pública. Ele converte todos os pacotes que vem de uma rede para o formato necessário para ele trafegar e ser roteado adequadamente em outra rede, sendo necessário reescrever os números de portas para os protocolos UDP e TCP.

À medida que os dados trefegam pelo roteador, este além de realizar a tradução dos endereçamentos em tempo real, vai gerando uma tabela de roteamento, baseando-se nos endereços e portas utilizados na comunicação entre o roteador e o nó externo. Assim, ao receber o retorno de uma requisição em uma determinada porta a partir de um determinado endereço de origem, o roteador é capaz de identificar qual nó interno que deverá receber o pacote.

Juntamente com o protocolo SIP (*Session Initiation Protocol*), o WebRTC tem problemas de NAT e Firewall. Neste caso, é altamente recomendado o uso dos protocolos a seguir.

[2]Criado como solução do esgotamento de endereços no formato IPv4. Seu endereço é formado por 128 bits, gerando um número de endereços possíveis muito alto.

2.2.1 *Interactive Connectivity Establishment (ICE)*

O ICE é um protocolo para travessia de NAT para *streams* de mídia, estabelecidas sob o modelo Oferta e Resposta. Surgiu com o intuito de servir como uma solução geral para diversas topologias de rede, resolvendo o problema de desenvolvedores e administradores de rede que precisam fazer suposições e estudo a respeito das topologias onde seus sistemas serão implantados (VARANDA, 2008).

O modelo de Oferta e Reposta apresenta algumas dificuldades em operar através de NAT, pois para estabelecer um fluxo de mídia entre dois usuários é trocado entre eles mensagens de endereços IP e portas, que não são tratados corretamente pelos dispositivos de NAT.

Para viabilizar o estabelecimento de conexões multimídia entre dois usuários, o protocolo ICE pode tirar proveito ou compor as funcionalidades de outros protocolos, neste caso o STUN e TURN, para solucionar o problema de trocas de mídia entre dois clientes que estejam por trás de NAT.(VARANDA, 2008)

Etapas do ICE:

- **Identificar endereços candidatos** - Antes mesmo de enviar ou receber uma oferta SDP(explicado posteriormente), são identificados os endereços canditados para cada fluxo multimídia, em cada um dos agentes que desejam se comunicar.
- **Avaliação e priorização dos candidatos** - Através de um algoritmo, são categorizados e priorizados os canditados em relação ao melhor caminho para a comunicação.
- **Envio dos endereços candidatos** - Os endereços são inclusos na oferta e resposta entre os agentes.
- **Checagem de conectividade** - Através do protocolo STUN, são feitos os testes de conectividade dos endereços canditados recebidos.
- **Estabelecimento da sessão** - O protocolo ICE envia uma oferta SDP a todos os candidatos com conectividade para um determinado fluxo multimídia, informando o melhor caminho a todos os agentes envolvidos, incluindo roteadores.
- **Manutenção da conexão** - Após o estabelecimento da conexão, o ICE usando alguns recursos do STUN, executa o processo de manutenção do fluxo de dados através de pacotes *Keep Alive* enviados em intervalos regulares.

2.2.2 *Session Traversal Utilities for NAT (STUN)*

Em sua primeira versão, definida no RFC 3489 com o nome de *Simple Travessar of UDP through NATs*, tinha como função identificar e caracterizar o tipo de NAT atribuído a um determinado agente e realizar sua travessia.

Com o tempo, ela apresentou algumas falhas em certos cenários e comportamentos de NAT, não se tornando uma solução geral o suficiente para a abordagem dos problemas apresentados. Surgiu então uma segunda versão, sendo especificada como *draft* no processo do IETF, e sendo intitulado de *Session Traversal Utilities for NAT*. Além de servir de base para outros protocolos, em especial o protocolo ICE, permite que um agente descubra qual porta e endereço IP externo está mapeado ao seu endereço IP e porta interno, ou seja, compõem o chamado endereço reflexivo. (VARANDA, 2008)

Em outras palavras, o STUN permite que um cliente obtenha um endereço publicamente acessível, a fim de estabelecer uma ligação direta. Exemplo na Figura 2.4.

O STUN possui limitações, fucionando somente com três tipos de NAT: *full cone NAT, restricted cone NAT, and port restricted cone NAT*. Nos casos de *restricted cone ou restricted cone*, o cliente deve enviar previamente um pacote para o ponto final, antes do NAT permitir que pacotes retornem do ponto final até este cliente. O STUN não funciona com NAT simétrico (também conhecido como NAT bi-direcional), que é encontrado frequentemente nas redes de grandes empresas.

Uma observação importante em relação a efetividade deste protocolo, segundo dados do webrtcstats[3], há chances de 86% de sucesso na ligação utilizando servidores STUN, podendo variar para menos, caso o cenário em questão seja muito complexo.

2.2.3 *Traversal Using Relays around NAT (TURN)*

Quando não há possibilidade de estabelecer um canal direto entre dois agentes posicionados atrás de um NAT, é necessário recorrer a um *host* intermediário, que irá agir como um *relay*. Este normalmente se encontra na rede pública e tem a função de retransmitir os pacotes de dados entre os agentes(VARANDA, 2008). Exemplo de utilização na Figura 2.5. (VARANDA, 2008)

O TURN é uma alternativa que consome muitos recursos do servidor STUN, como processamento, largura de banda, entre outros. Por isso é recomendado como a última alternativa para

[3] webrtcstats.com

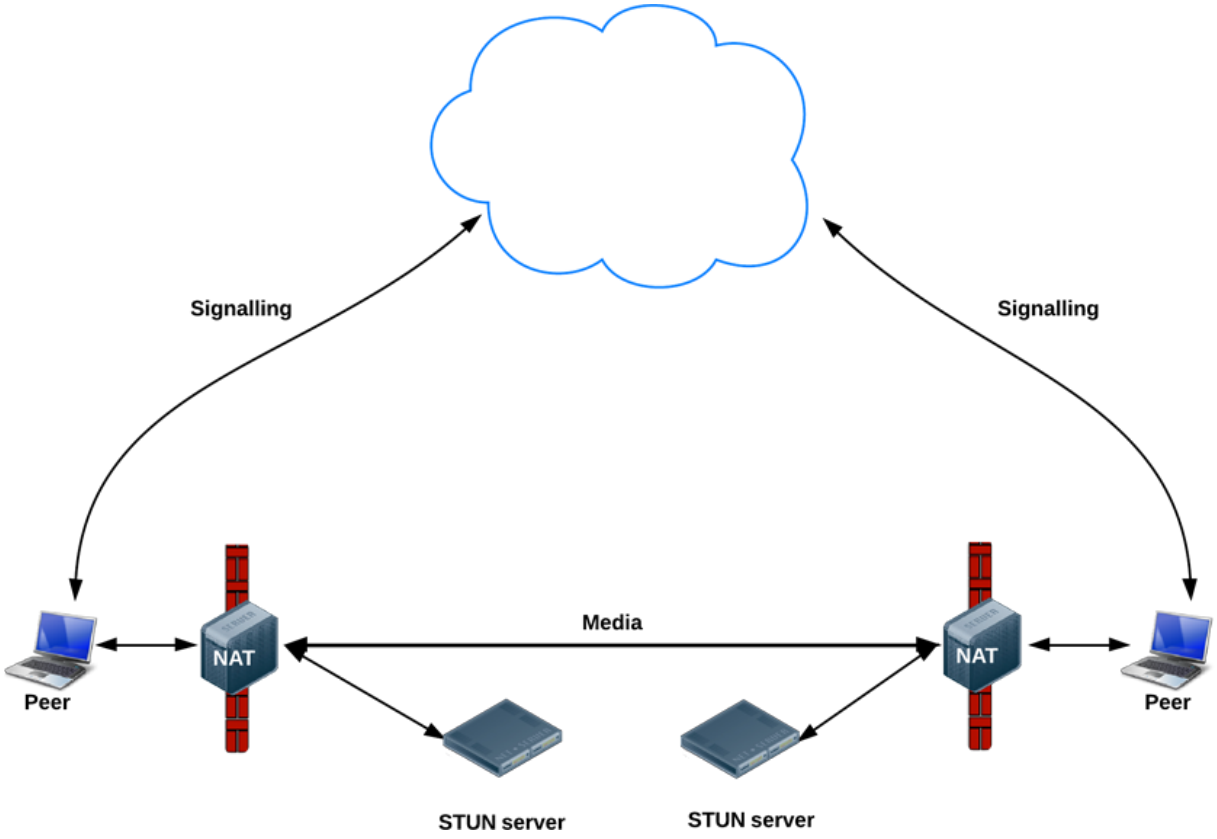

Figura 2.4: Cenário utilizando o protocolo STUN.

Fonte: html5rocks.com, 2012

viabilizar a comunicação entre dois agentes. (VARANDA, 2008)

2.3 Sinalização

Para estabelecer uma chamada entre os usuários do serviço, é necessário um mecanismo de sinalização para negociação de parâmetros da sessão. No WebRTC, a sinalização dos recursos (áudio, vídeo ou dados) fica a critério do próprio desenvolvedor da aplicação. É ele quem define a sinalização mais adequada ao projeto. Neste trabalho foi utilizado o SIP (*Session Initiation Protocol*).

2.3.1 SIP

O SIP é um protocolo para estabelecimento de chamadas, manutenção e finalização de sessões multimídia. Foi proposto pela IETF (Internet Engineering Task Force), RFC 3261. Sua concepção foi baseada nos protocolos HTTP (*Hypertext Transfer Protocol*) e SMTP (*Simple Mail Transfer Protocol*) (mensagens em texto puro). Habilidades do SIP:

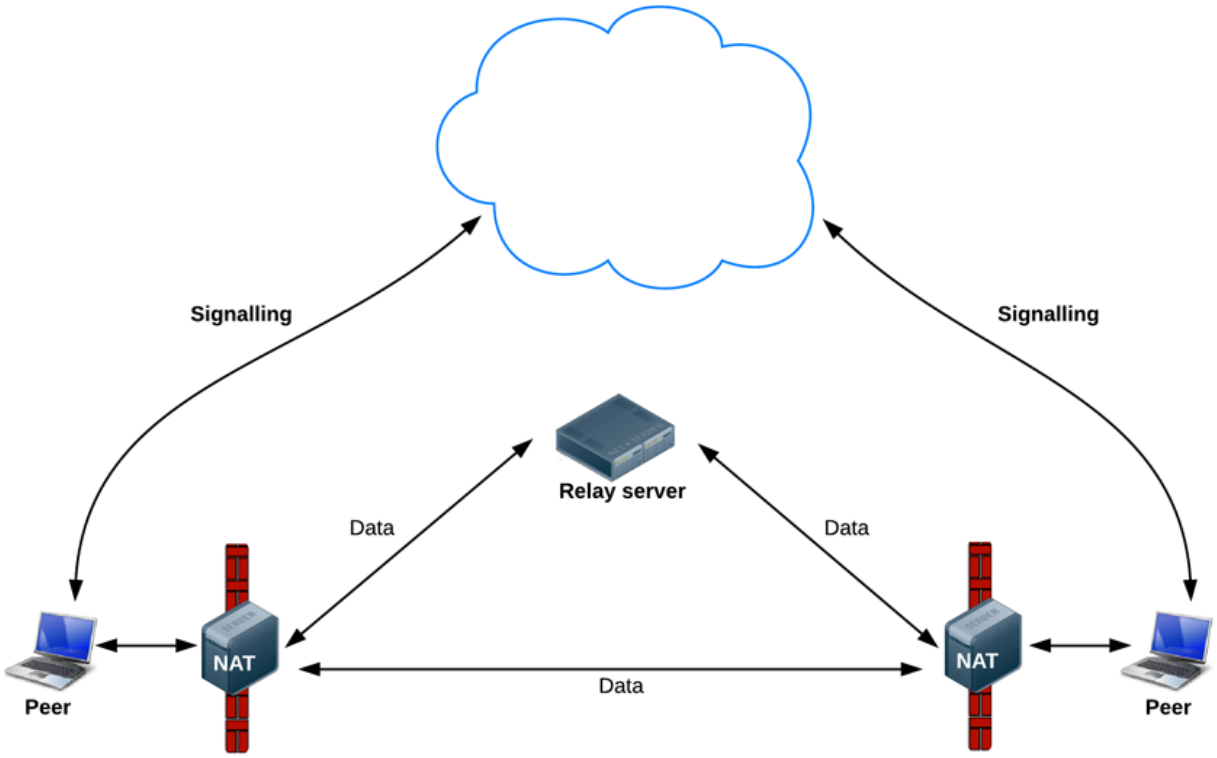

Figura 2.5: Cenário utilizando o protocolo TURN.

Fonte: html5rocks.com, 2012

- Localização do usuário
  - Determina o atual dispositivo do usuário
- Disponibilidade do usuário
  - Indica se o usuário está disponível para uma comunicação
- Habilidades do usuário
  - Determina quais os codecs o usuário possui
- Estabelecimento de sessão
  - Determina parâmetros como as portas usadas
- Gerenciamento de sessão
  - Transferência de sessão e modificação dos parâmetros

Uma rede SIP é composta por entidades SIP lógicas, na qual possuem papeis distintos, podendo ser clientes (originam pedidos), servidores(recebem pedidos) ou ambos. As entidades logicas são:

- Agente Usuário (*User Agente - UA*): Trata-se de um ponto ponto final de uma comunicação SIP (Telefones IP, *softphones*, ATAs). São formado por clientes (UAC) que originam pedidos de conexão e servidores (UAS) que recebem e respondem a estes pedidos.
- Servidor Proxy (*Proxy Server*): É a entidade intermediária que atua como cliente e servidor, com o objetivo de originar pedidos em nome de outros clientes. Pode interpretar os pedidos, reescrever e encaminhá-los caso haja necessidade.
- Servidor de Redirecionamento (*Redirect Server*) : Aceita pedidos SIP e faz a correspondência do endereço destino com os endereços finais.
- Servidor de Registro (*Registrar Server*): Geralmente usado em conjunto com o Servidor Proxy e o Servidor de Redirecionamento, possibilita o registro dos usuário e sua localização na rede.

Para iniciar uma sessão SIP, um usuário envia uma mensagem de INVITE com todos os dados necessários para o início de sessão no cabeçalho. Caso o outro usuário aceite o INVITE, este responderá com uma mensagem de 200 OK. Na Figura 2.6 um exemplo de uma chamada SIP.

2.3.2 *WebSockets*

A *web* tem sido construída com base no mecanismo de pedido e resposta de HTTP, ou seja, quando um usuário acessa uma página, nada irá acontecer até que ele clique em outra página para atualizar as informações desta. Mais tarde, o AJAX veio para deixar a *web* mais dinâmica, porém, toda a comunicação HTTP continuava sendo direcionada pelo cliente, o que ainda exigia interação do usuário ou temporizador que atualizasse a página. (SOUTO. . . , 2013)

A tecnologia que permite o servidor enviar dados atualizados para os clientes, e que foi utilizado por algum tempo, é conhecido como sondagem longa. Um dos principais problemas deste tipo de solução era o *overhead*de HTTP, que poderia prejudicar soluções que não são tolerantes a atrasos, como jogos *online*. Para solucionar este problema, surgiu então o WebSocket.

WebSocket é uma tecnologia que permite a comunicação full-duplex sobre um único *socket* TCP, entre cliente e servidor *web*. Ou seja, há uma conexão persistente onde ambas as partes podem começar a enviar dados a qualquer momento, diferente do método *Ajax Long Polling*, que permite que a conexão fique aberta, aguardando uma resposta do servidor para que possa ser fechada e, depois, reaberta. Ou também do método *Long Polling* que fica enviando requisições, mesmo que o servidor alegue, consecutivamente, não ter nenhuma resposta.

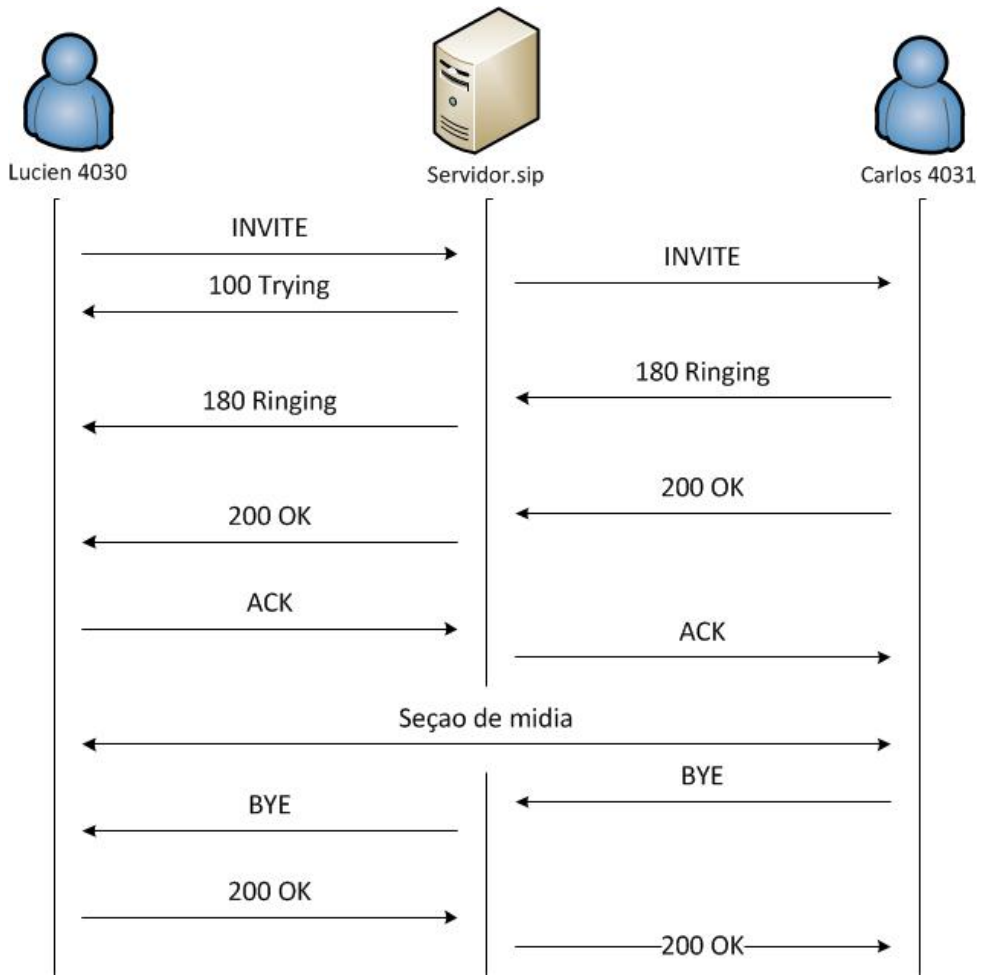

Figura 2.6: Exemplo de chamada SIP.

2.3.3 SIP *over WebSockets*

SIP over Websockets é uma especificação que define um subprotocolo de Websocket, permitindo a troca de mensagens SIP entre um cliente e um servidor *web*. Assim como o SIP, ela é composta por algumas entidades (AMARAL, 2013). As duas principais são:

- *SIP WebSocket Client*: entidade responsável por abrir ligações de *websockets* de saída para o servidor e que se comunica por *WebSocket SIP subprotocol*.
- *SIP WebSocket Server*: responsável por escutar ligações de entrada dos usuários e que se comunica por *WebSocket SIP subprotocol*.

2.4 Voz sobre IP (VoIP)

O VoIP tem como objetivo prover uma forma alternativa aos sistemas tradicionais de telefonia, tentando manter as mesmas funcionalidades e qualidade, aproveitando-se da Internet

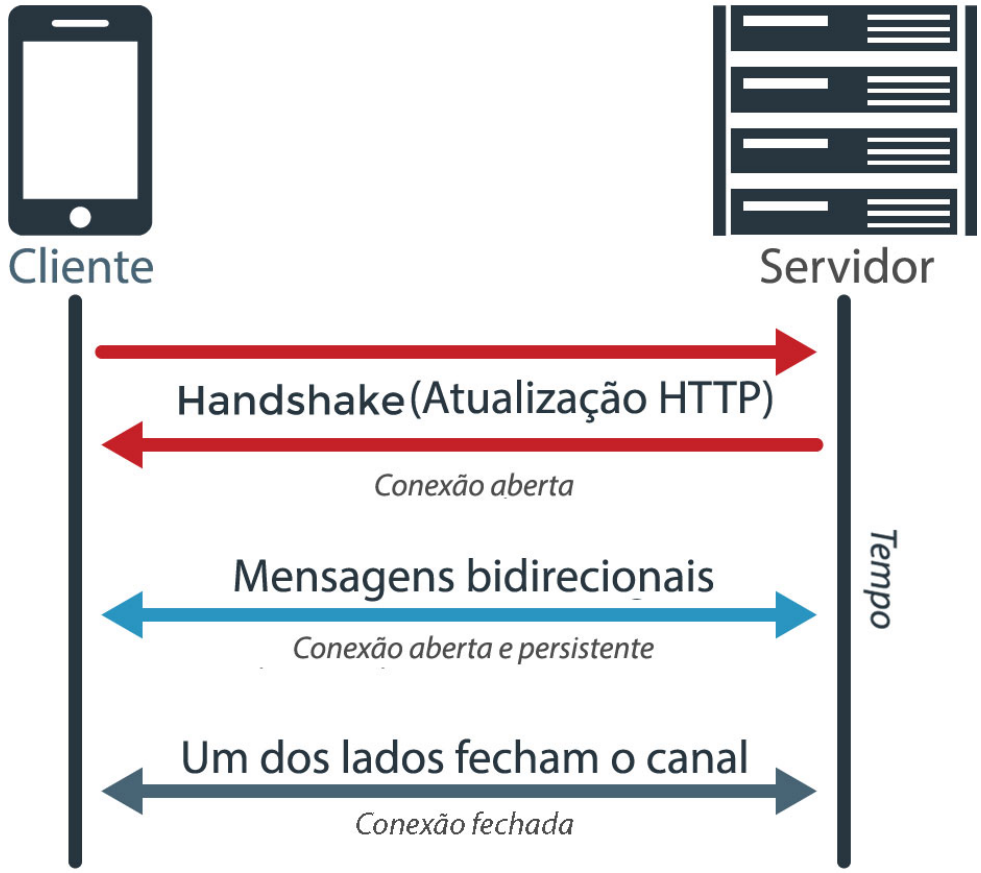

Figura 2.7: Funcionamento básico do WebSocket.

Fonte: pubnub.com, 2015

para o transporte de voz e dados. A *International Telecommunication Union* (ITU) e *Internet Engineering Task Force* (IETF) desenvolveram um conjunto de novos padrões e protocolos para viabilizar a comunicação com as mesmas características das redes tradicionais.

2.4.1 Protocolos essenciais

O protocolo SIP, mencionado na subseção 2.3.1, trabalha juntamente com outros protocolos que auxiliam a troca de dados entre os participantes de uma chamada. Estes serão abordados logo abaixo, de maneira direta, pois o intuido aqui é mostrar o funcionamento básico de uma chamada, e não o funcionamento aprofundado de cada protocolo.

Real-Time Transport Protocol (RTP)

O RTP é um protocolo utilizado para o transporte de mídias contínuas de tempo real, como áudio, vídeo ou dados, via Internet. Foi definido inicialmente em 1996 pela RFC 1889, sendo substituído pela RFC 3550, em 2003. Foi desenvolvido podendo atuar sobre o UDP, não sendo

sensíveis a perda, porém em relação a atrasos de envio e recebimento de dados, possuem alguns mecanismos que auxiliam na qualidade da transmissão.

Os principais serviços providos pelo RTP são:

- Identificação do tipo de *payload*;
- Sequenciamento de pacotes - Otimizar a reconstrução dos fluxos multimídia, que podem desordenar ao longo do caminho;
- Marcação temporal - Permitir a sincronização e o cálculo da variação de atraso;
- Monitoramento da entrega.

O RTP não define um conjunto de portas espefícicas para o tráfego da mídia. O único padrão que ele segue é de alocar portas pares para o protocolo UDP e alocar a porta subsequente para o protocolo de controle RTCP (*Real-Time Transport Control Protocol*). Por não definir exatamente o intervalo de porta a ser utilizado nesse tráfego, acaba tornando-o sensível em relação à travessia de NAT e Firewall em geral. Isso porque não há possibilidade de encaminhamento de dados, se não houver conhecimento exato de qual porta que cada cliente está utilizando. Para o tratamento do comportamento deste protocolo, há alguns metodos de NAT já existentes, como foi mencionado na subseção 2.2.

Mesmo possuindo todos estes mecanismos, o RTP não provê garantias de Qualidade de Serviço, sendo necessário fazer uso de outro protocolo, o RTCP.

Real-Time Transport Control Protocol (RTCP)

O RTCP é utilizado para monitorar a qualidade do serviço e para transportar informações dos participantes de uma sessão RTP em andamento. É um protocolo de apoio ao RTP.

Principais funções:

- Retorno sobre o desempenho da aplicação e da rede - Útil para adaptar a taxa de transmissão para balancear o desempenho e/ou qualidade da mídia;
- Informações para sincronização de fluxos de áudio e vídeo;
- Transportar um identificador de nível de transporte.

Secure Real-time Transport Protocol (SRTP)

O SRTP é um protocolo obrigatório quando falamos sobre fluxo de mídia do WebRTC. Foi desenvolvido por um grupo de especialistas em criptografia das organizações Cisco e Ericsson, onde em 2004 teve sua publicação na IETF, norma RFC 3711. Ele tem como objetivo oferecer confiabilidade dos dados transportados, autenticação e proteção contra ataques de reprodução da carga útil dos protocolos RTP e RTCP.

Utiliza o sistema AES como codificador/decodificador padrão da mídia, um sistema validado e definido como padrão criptográfico em 2001 pelo Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST). (ALMEIDA, 2014) Mesmo o SRTP tendo um papel importante na segurança dos dados, segundo a norma, seu uso não é obrigatório, podendo ser desabilitado caso necessário.

Session description protocol (SDP)

Documentado pela primeira vez na RFC 2327 em abril de 1998, atualmente está em sua segunda versão desde julho de 2006, na RFC 4566. Como sua utilização está sempre em conjunto com um protocolo de transporte, não possui mecanismo de transporte próprios.

Utilizado normalmente com o SIP e RTP, o SDP tem como principal finalidade descrever as características do fluxo multimídia a ser estabelecido entre os dois agentes, informando o tipo de mídia a ser transportada, formatos de codificação, protocolos de comunicação e portas a serem utilizadas no estabelecimento da comunicação. Ou seja, é um protocolo com parâmetros de configuração, para iniciação e manutenção de sessões.

Suas mensagens são baseadas em texto, permitindo que seja lida e analisada de forma simples e direta.

2.5 Asterisk

O Asterisk é um *software* de código aberto, que implementa os recursos encontrados em um PABX convencional, utilizando a tecnologia VoIP. Foi desenvolvido e ainda é mantido pela empresa Digium (surgida em 1999). Atualmente, diversas empresas de pequeno a grande porte tem adotado este tipo de solução, pois permite o desenvolvimento de multiprotocolos a aplicações de comunicações em tempo real com voz e vídeo. Por ser código aberto, o *software* pode ser modelado de acordo com a necessidade da empresa.

O suporte ao WebRTC foi adicionado ao Asterisk em sua versão 11, sendo criado o módulo res_http_websocket que permite programaddores Javascript desenvolverem soluções em que haja interação e comunicação do WebRTC com o Asterisk. Dentro do módulo chan_sip, foram adicionados WebSockets para permitir o uso de SIP como protocolo de sinalização.

2.6 Tecnologias auxiliares

Atualmente já existem algumas soluções WebRTC com funcionalidades diferentes que podem ser usadas e servir de base para outras aplicações. Uma delas é o sipML5.

2.6.1 sipML5

O sipML5 é uma aplicacao desenvolvida pela Doubango Telecom, que implementa a pilha do protocolo SIP em Javascript. Pode ser usado em qualquer *web browser*, para uma ligação a uma rede SIP permitindo realizar e receber chamadas de vídeo ou voz. Alguma das funções suportadas:

- chamadas de vídeo e áudio;
- mensagens instantâneas;
- presença;
- transferência de chamadas;
- chamada em espera;

Seu uso, é recomendável no Google Chrome e Firefox *stable*. Nos demais navegadores, é necessário instalar a extensão *webrtc4all*. Vale ressaltar que essa informação pode sofrer alterações com o tempo. Exemplo na Figura 2.8.

É importante ressaltar que, a partir da versão 11, o Asterisk tem suporte nativo ao WebRTC, dispensando o uso de um gateway como o *WebRTC2SIP* que realiza conversões dos pacotes enviados pelo *endpoint* que utiliza webRTC, no caso o sipML5, para o PABX IP.

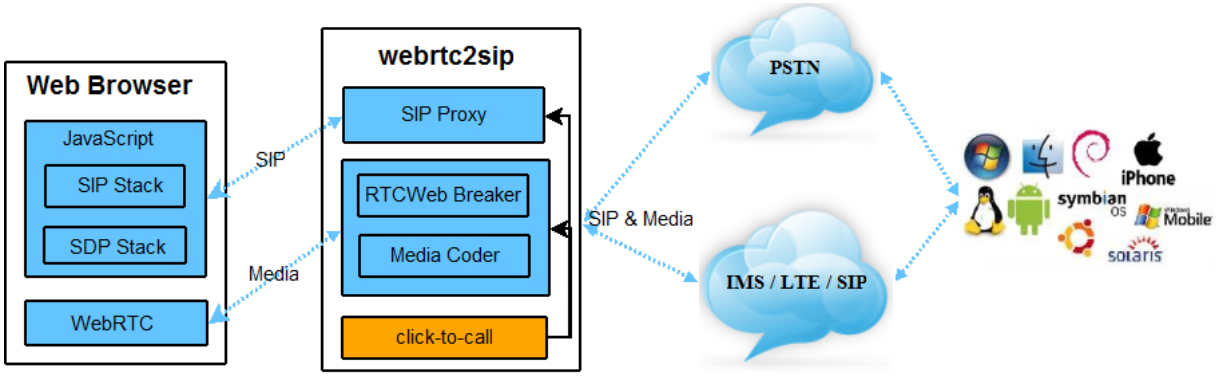

Figura 2.8: Uso do sipML5

Fonte: sipml5.com, 2012

2.6.2 Node.js

O Node.js é uma plataforma orientada a eventos, do lado do servidor, baseada no V8 Javascript Engine[4], com o objetivo de criar aplicações de redes rápidas e escaláveis. Servidores que utilizam a linguagem Java ou PHP que possui muitos acessos, necessitam de uma grande quantidade de memória, pois a cada conexão, é alocada cerca de 64KB a 32MB do sistema, necessitando de uma grande quantidade de memória . O Node.js resolve esta questão trocando a maneira como cada conexão é tratada no servidor. Cada conexão cria um processo que não requer que o bloco de memória o acompanhe. O Node.js não permite bloqueios e alega que pode suportar dezenas de milhares de conexões simultâneas.

Integrado ao Node.js, o NPM *Node Package Manager*, implementa um repositório de módulos, onde cada um possui uma funcionalidade particular. Projetos utilizando Node.js, podem fazer uso desse sistema. Neste trabalho, será utilizado apenas o Node Package Manager em nosso projeto, pois ele disponibiliza um módulo essencial para o projeto, o Socket.io, descrito adiante.

2.6.3 Sails.js

Sails.js é um *framework* que utiliza o conceito MVC[5], com suporte para os requisitos de aplicativos modernos: APIs orientadas a dados com uma arquitetura orientada a serviços escaláveis. Indicados para construção de serviços em tempo real.

O Sails.js roda sobre o Node.js, e fornece toda estrutura de diretórios e arquivos que poderão

[4]V8 Javascript Engine: É o interpretador de Javascript *open source* implementado pelo Google em C++ e utilizado pelo Google Chrome.

[5]*Model-View-Controller* é um padrão de arquitetura de *software* que separa a informação (e as suas regras de negócio) da interface com a qual o usuário interage.

ser utilizados (tudo de maneira organizada), não havendo necessidade de criar um projeto do início . Entrega também algumas dependências que será necessário para capturar alguns dados do Asterisk. A principal delas é o Socket.io.

2.6.4 Socket.io

O Socket.io permite realizar comunicação bidirecional utilizando uma das APIs de transporte na seguinte ordem: WebSockets, FlashSockets, AJAX long polling, AJAX multipart streaming, Forever Iframe ou JSONP Polling. O motivo dessa ordem é garantir compatibilidade *cross-browser*[6].

Largamente utilizado para geração de relatórios em tempo real, *chats*, jogos *multiplayer online*, aplicações de interação em tempo real com diversos usuários. A Figura 2.9 mostra um exemplo básico de um usuário se registrando em um sistema utilizando Socket.io.

É através do Socket.io que iremos implementar o sistema de *chat* e coletar informações do Asterisk para a atualização das informações em nosso sistema.

2.6.5 Asterisk Manager Interface (AMI)

O AMI permite uma aplicação se conectar em uma instância do Asterisk e executar comandos ou ler eventos do PBX através de uma conexão TCP/IP . Desenvolvedores e integradores podem ter inúmeras possibilidades atraves desta API, podendo por exemplo controlar o estado de um usuário, direcionando este a regras personalizadas. As figuras abaixo (2.10, 2.11) mostram como cliente e servidor se comunicam para a troca de informações.

Atualmente, há diversas linguagens que conseguem realizar este tipo de conexão com a API do Asterisk, como o Javascript (descrito adiante), PHP, Java, Phyton, C++, entre outras.

2.6.6 NodeJS Asterisk Manager

É um módulo do Node.js criado pela própria comunidade de desenvolvedores *web*, que permite interação de alguma aplicação rodando sobre o Node.js com um servidor Asterisk. Através desta API, é aberto um canal de comunicação com o Asterisk, no qual é possível ficar escutando os eventos (*Asterisk AMI Events*) que estão ocorrendo no momento, e/ou enviar alguns comandos (*Asterisk AMI Actions*), para obter alguma resposta. Com isto, é possível capturar os ramais

[6]Cross-browser refere-se à habilidade de um site, Aplicação Web, contructor HTML ou script side-client suportar múltiplos navegadores.

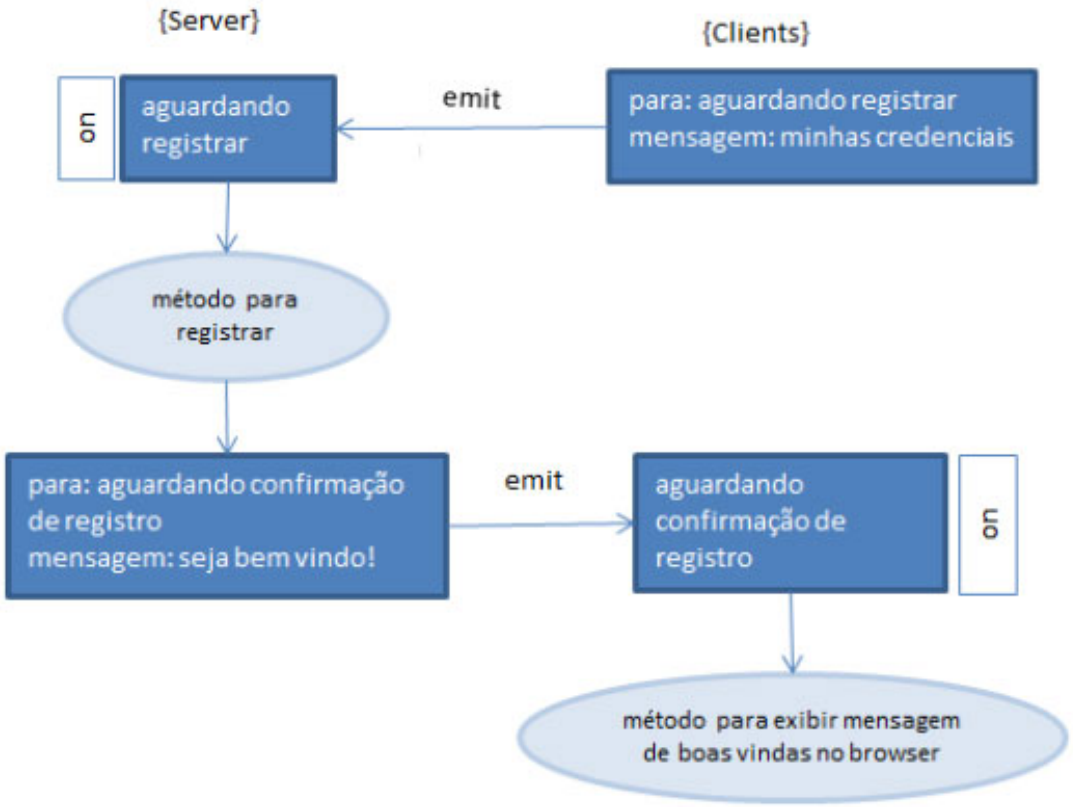

Figura 2.9: Funcionamento básico do Socket.io

Fonte: Venancio, 2013

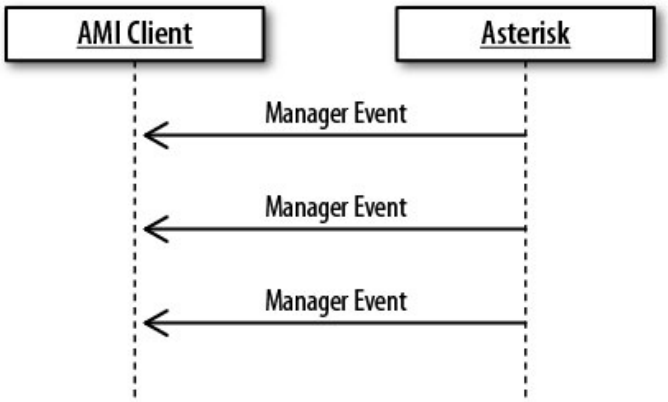

Figura 2.10: Cliente escutando eventos do Asterisk

Fonte: asteriskdocs.org, 2012

registrados, estado dos ramais, o IP de cada participante, entre outras informações não somente dos ramais, mas também dos canais de transmissão.

No Apêndice C, é mostrado uma lista de eventos e acões que podem ser utilizados na captura de alguma informação do Asterisk.

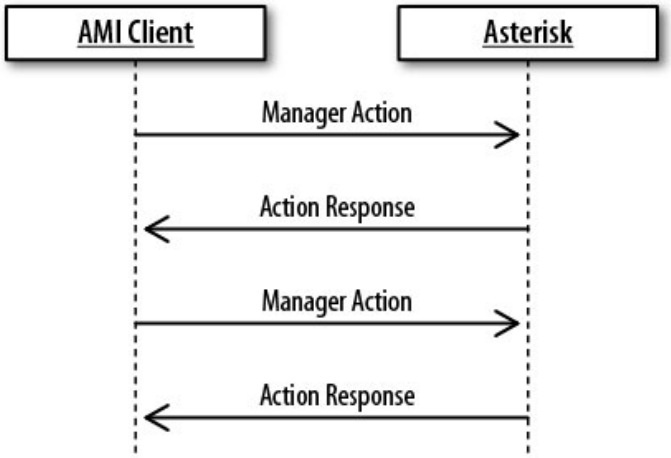

Figura 2.11: Cliente enviando uma requisição ao Asterisk

Fonte: asteriskdocs.org, 2012

2.7 Linguagens utilizadas no projeto

2.7.1 HTML5

Criada por Tim Berners Lee na década de 1990, e com suas especificações controladas pela W3C, o HTML (*HyperText Markup Language*) é uma linguagem de marcação utilizada para produção de páginas na *web*, que permite a criação de documentos que podem ser lidos em praticamente qualquer navegador.

Para escrever documentos HTML é necessário de apenas um editor de texto e conhecimento sobre a linguagem. Esta são compostas por *tags* servem para indicar a função de cada elemento da página *web*. Os tags funcionam como comandos de formatação de textos, formulários, *links*, imagens, tabelas, entre outros.

O HTML versão 5 adiciona várias novas funções sintáticas, incluindo as *tags* de vídeo e áudio que são indispensáveis para o funcionamento o WebRTC. Na Figura 2.12 um simples exemplo utilizando as principais tags do HTML5.

2.7.2 CSS

O CSS *Cascading Style Sheets* é uma linguagem para estilos que define o *layout* de documentos HTML, ou seja, ela controla os tipos de fontes, cores, alturas e larguras, posicionamento, entre outros elementos de uma página *web*. Foram criadas basicamente para deixá-las mais elegantes e atrativas aos usuários.

```
<video controls>
  <source src="foo.ogg" type="video/ogg; codecs=dirac, speex">
  Seu navegador não suporta o elemento <code>video</code>.
</video>

<audio src="audio.ogg" controls autoplay loop>
<p>Seu navegador não suporta o elemento audio </p>
</audio>
```

Figura 2.12: Utilização de vídeo e áudio em uma página *web*.

Fonte: Togo, 2015

2.7.3 Javascript

Criada por Brendan Eich e lançado pela primeira vez na versão beta do navegador NetScape 2.0 em 1995, o Javascript é uma linguagem programação interpretada no lado cliente, ou seja, é processada pelo próprio navegador. Com o JavaScript podemos criar efeitos especiais para nossas páginas na *Web*, além de proporcionar uma maior interatividade com nossos usuários (JAVASCRIPT, 2014). Hoje o Javascript é considerado a principal linguagem para programação *client-side* em navegadores *web* e todos os exemplos de códigos aqui apresentados, o utilizam.

3 *O sistema proposto*

Este capítulo apresenta uma visão geral do sistema proposto, descrevendo uma sequência de etapas que guiaram a implementação do projeto.

3.1 Descrição geral do sistema

O sistema tem como objetivo realizar a interação de um *softphone web*, tendo como base a tecnologia WebRTC, com uma central IP (neste cenário, o Asterisk). Além disso, este *softphone* sofrerá algumas adaptações, mostrando como o WebRTC pode ser útil em futuras aplicações, e o quão robusta esta solução pode ser. Ou seja, a idéia é desenvolver um *softphone web*, com outras funcionalidades que sejam útil ao usuário.

A versão do Asterisk que deverá ser utilizado é a 11 ou superior, pois já dispõe nativamente do suporte ao WebRTC. Na parte do WebServer que rodará o servidor Apache, será utilizado o sipML5, servindo como base da aplicação a ser implementada, pois este já implementa a pilha do protocolo SIP, como foi explicado na seção 2.6.1. Em relação aos computadores, recomenda-se fortemente a utilização dos navegadores Google Chrome, Mozilla Firefox e Opera, que oferecem uma boa compatibilidade com a tecnologia WebRTC. A Figura 3.1 mostra um cenário que pode ser montado com estas tecnologias.

3.2 Levantamento de requisitos

Após algumas pesquisas envolvendo projetos utilizando WebRTC, iniciou-se a fase de desenvolvimento do projeto. Foram definidos os requisitos para que esta fosse implementada de acordo com o pretendido.

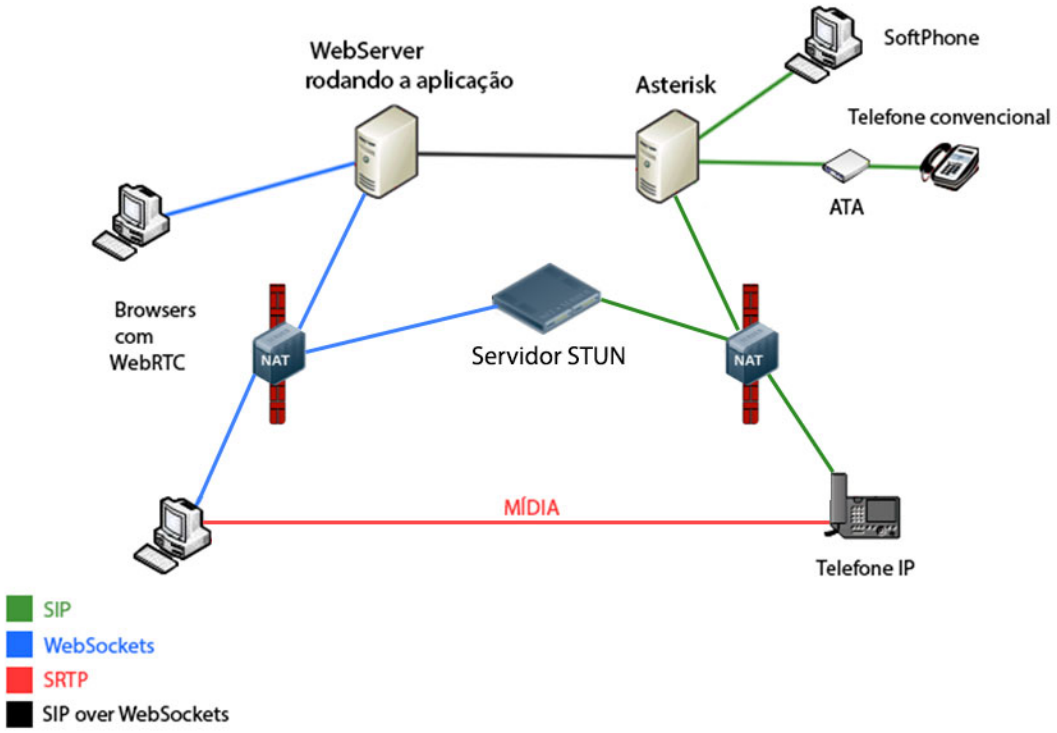

Figura 3.1: Cenário representativo de testes

Fonte: Togo, 2015

3.2.1 Requisitos Funcionais

O objetivo deste projeto é desenvolver uma aplicação WebRTC/SIP robusta, onde um usuário consegue interagir com outros de maneira prática, e mostrar o desempenho da aplicação em diferentes plataformas. Esta seção apresenta os requisitos que foram definidos inicialmente no projeto.

Estabelecimento de chamada

O sistema deve permitir que um usuário consiga estabelecer uma chamada através de vídeo, áudio ou ambos, com outro usuário que estiver conectado no sistema. No caso onde os ramais conectados estejam em ligação, ausente ou não conectados o sistema deverá informar de alguma maneira o porquê da ligação não ter sido completada.

Informações da chamada

Durante a conversação entre os ramais, o sistema deve informar quem são os ramais que estão participando da chamada e o tempo de conversação entre estes, tudo em tempo real.

Estado de presença e Lista de contato

É um serviço que se encontra na maior parte de aplicações de *chat*. É necessário um banco de dados que contenha todas as informações dos usuários que se encontram na lista de contatos para, quando necessário, exibir o estado de presença que o ramal se encontra (Ocupado, Disponível, Ausente, Desconectado). Sempre que o estado de um usuário for alterado, é preciso notificar em tempo real todos os outros ramais conectados.

***Chat*/mensagem instantâneas**

Outro item desejável no sistema é um serviço de *chat*. Cada usuário poderá abrir um canal de comunicação com outro ramal, enviando mensagens instantâneas.

3.2.2 Requisitos Não Funcionais

Esta seção está relacionada ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, tecnologias envolvidas. Será descrito o que se deve esperar do sistema desenvolvido.

Desempenho

- Deseja-se que as chamadas e a conversação não tenham um atraso superior a dois segundos.
- Toda a lista de ramais deverão ser atualizados em tempo real.
- O sistema de *chat* deverá funcionar em tempo real, independente da quantidade de janelas que estejam abertas.

Usabilidade

A interface deverá ser intuitiva. Desde a localização de um usuário disponível no sistema até a realização e encerramento da chamada.

Segurança

- Nenhum agente externo poderá ter acesso aos dados trafegados.
- Cada ramal deverá possuir uma senha única de acesso a aplicação/registro no Asterisk.

Restrições

É garantido funcionamento apenas para versões atuais do Google Chrome, Mozilla Firefox e Opera.

3.3 Casos de Uso

Os diagramas de Casos de Uso têm o objetivo de auxiliar a comunicação entre os analistas e o cliente, descrevendo um cenário que mostra as funcionalidades do sistema do ponto de vista do usuário. As principais funcionalidades de seu sistema podem ser vistas na Figura 3.2.

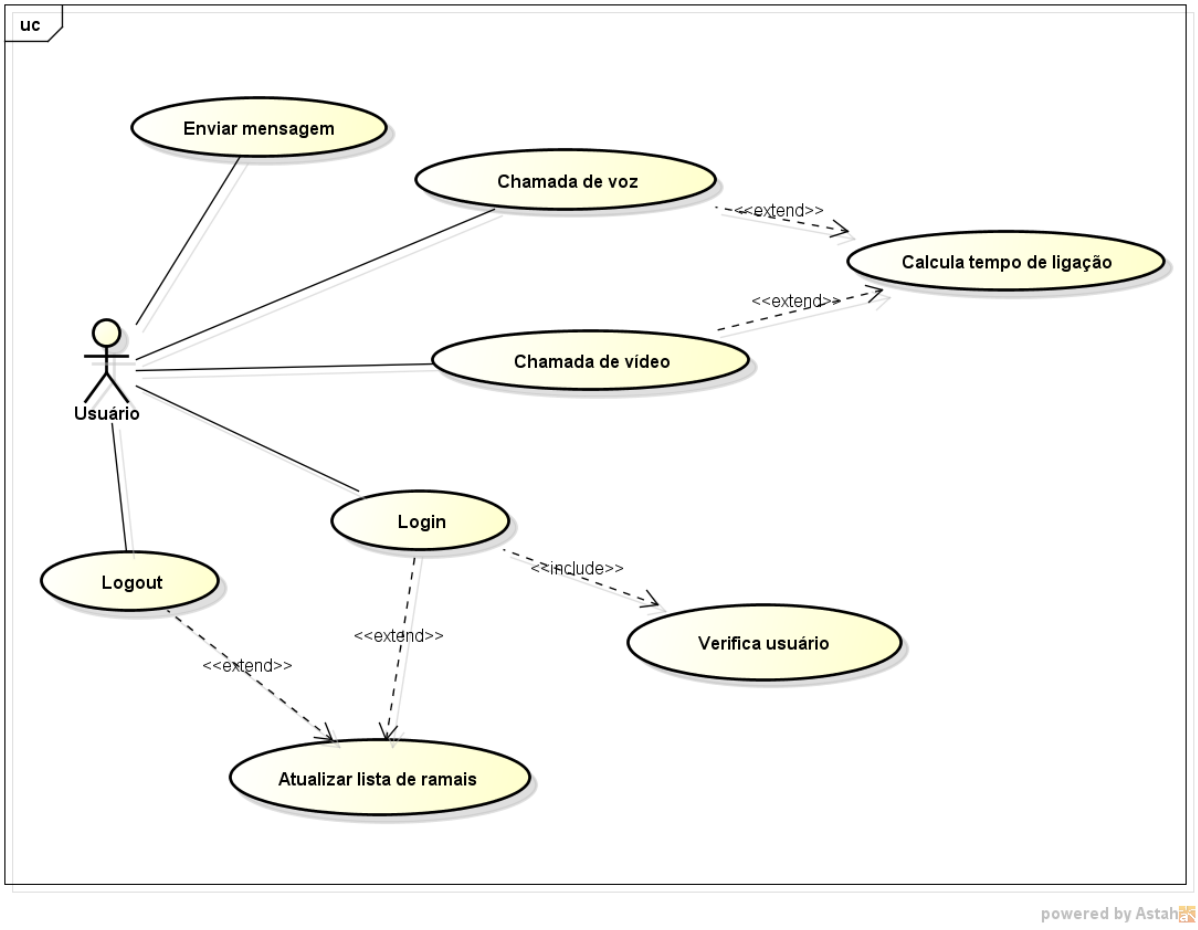

Figura 3.2: Diagrama de Casos de Uso.

Fonte: Togo, 2015

O diagrama UML de Casos de Uso é constituído por:

- Atores: representado por um boneco e um rótulo com o nome do ator. Este pode ser um usuário humano ou um outro sistema computacional.

- Caso de uso: representado por uma elipse e um rótulo com o nome do caso de uso. Ela define uma funcionalidade do sistema. Uma função pode ser estruturada em outras funções e, portanto, um caso de uso pode ser estruturado.
- Relacionamentos: representado por setas. Ajudam a descrever quem está relacionado a quem no sistema.

Os casos de uso deste sistema são apresentados nas tabelas 3.1 a 3.4.

Caso de Uso	Login.
Descrição	O usuário preenche todos os campos, informando o número do seu ramal e senha respectiva. Caso tudo estiver de acordo, é liberado a tela de bloqueio do sistema.
Atores	Usuário.
Précondições	O usuário deve estar devidamente registrado no arquivo sip.conf do Asterisk
Fluxo principal	1. Usuário acessa ta tela de login 2. Usuário preenche todos os campos, informando o número do ramal, senha e servidor. 3. O sistema verifica se o usuário e senha estão de acordo. 4. O usuário é liberado para acessar o sistema
Fluxo de exceção	2.1 Preenche o usuário e senha com dados incorretos. 2.2 O sistema apresenta uma mensagem de erro.

Tabela 3.1: Efetuando login na aplicação.

Todos estes casos de uso, servem como base para a identificação de parte do sistema, e quais funcionalidades cada uma ficará encarregada.

3.4 Modelo do sistema *web*

Após levantado alguns exemplos de casos de uso, foi possível identificar quais funcionalidades o sistema deveria possuir, como estes ficariam interligados e como cada um iria se comportar. Para maior entendimento do sistema, serão apresentados diagramas de classe e sequencia mais relevantes.

Caso de Uso	Realizar chamada de voz.
Descrição	O usuário clica no ícone de microfone, que deve aparecer ao lado do ramal, para iniciar conversação.
Atores	Usuário.
Précondições	O ramal deve atender o Caso de Uso - Login, com sucesso.
Fluxo principal	1. Usuário checa todos os ramais disponíveis na lista de ramais. 2. Usuário clica no botão com ícone de microfone para realizar chamada com um determinado ramal. 3. O *browser* pergunta se o usuário permite a liberação da captura de áudio. 4. Usuário concorda. 5. O sistema tenta estabelecer chamada com o usuário destino. 6. É aberto um canal de comunicações entre os dois ramais. 7. Usuários conversam. 8. Um dos usuários clica no botão para encerar chamada. 9. O sistema se encarrega de enviar o sinal de desligamento para outra ponta.
Fluxo de exceção	3.1 Ramal destino está desconectado. 3.2 O sistema apresenta uma mensagem informando que o ramal destino não foi encontrado.

Tabela 3.2: Efetuando uma chamada com outro ramal.

3.4.1 Diagrama de classes

A Figura 3.3 apresenta um diagrama de classes do sistema. O sistema foi dividido em três grandes blocos. O bloco *front-end*, que fica encarregado de apresentar todas as informações que serão visualizadas pelos usuários, o bloco *back-end*, responsável em tratar todas as requisições vindas do *front*, e que trabalhará em conjunto com o Asterisk e um bloco do sipML5, que estará integrado com a aplicação que será desenvolvida.

Caso de Uso	Realizar chamada de voz.
Descrição	O usuário clica no ícone de câmera, que deve aparecer ao lado do ramal, para iniciar conversação.
Atores	Usuário.
Précondições	O ramal deve atender o Caso de Uso - Login, com sucesso.
Fluxo principal	1. Usuário checa todos os ramais disponíveis na lista de ramais. 2. Usuário clica no botão com ícone de câmera, para realizar chamada com um determinado ramal. 3. O *browser* pergunta se o usuário permite a liberação da captura de áudio. 4. Usuário concorda. 5. O sistema tenta estabelecer chamada com o usuário destino. 6. É aberto um canal de comunicações entre os dois ramais. 7. Usuários conversam. 8. Um dos usuários clica no botão para encerar chamada. 9. O sistema se encarrega de enviar o sinal de desligamento para outra ponta.
Fluxo de exceção	3.1 Ramal destino está desconectado. 3.2 O sistema apresenta uma mensagem informando que o ramal destino não foi encontrado.

Tabela 3.3: Efetuando uma chamada de vídeo com outro ramal.

Bootstrap

Ao subir o servidor da aplicação, todas as funções que fazem parte deste objeto serão executadas apenas. Este objeto abre um *socket* com o Asterisk para coletar informações dos usuários conectados, ligações, canais, etc. É utilizado para ouvir todos os eventos que estejam ocorrendo no Asterisk.

O Bootstrap também é responsável por fazer a temporização das ligações. Ao ser disparado o evento *Bridge*, o Node.js Asterisk Manager coleta todas as informações sobre o evento, incluindo os ramais participantes. O sistema trata estas informações e expõem ao usuário em

Caso de Uso	Enviar mensagem/*chat*.
Descrição	O usuário clica no ícone de *chat*, para enviar uma mensagem a um usuário.
Atores	Usuário.
Précondições	O ramal deve atender o Caso de Uso - Login, com sucesso.
Fluxo principal	1. Usuário checa todos os ramais disponíveis na lista de ramais. 2. Usuário clica no botão com ícone de balão de fala, para abrir uma janela de *chat* com um determinado ramal. 3. Usuário digita alguma mensagem no campo de escrita, e clica em enviar. 4. O sistema verifica para qual ramal que foi aberto a janela de *chat*, e se encarrega de enviar a mensagem. 5. Usuário recebe um alerta, na lista de usuários, informando que recebeu uma resposta. 6. Usuário lê a resposta.

Tabela 3.4: Enviando mensagens através do *chat*.

forma de dados.

Em linhas gerais, o Bootstrap ao ser inicializado, é responsável por abrir a conexão via *socket* com o Asterisk, onde ficará escutando todos os eventos que foram específicados no código e por colher todas as informações que possam ser úteis ao usuário. Estas informações então são enviadas ao *front-end*, que fará uso de alguma maneira.

SocketController

Enquanto o Bootstrap é responsável por ouvir todos os eventos do Asterisk, o SocketController é responsável por enviar *actions*. Ou seja, são enviadas requisições ao Asterisk solicitando alguma informação, onde a resposta será enviada de volta ao *back-end* da aplicação, que também enviará ao *front-end*.

Partes do funcionamento do *chat*, também ficam sob responsabilidade do SocketController, sendo encarregado pelo encaminhamento das mensagens. Transporta informações sobre quem está enviando a mensagem, o conteúdo da mensagem, e para quem esta se destina.

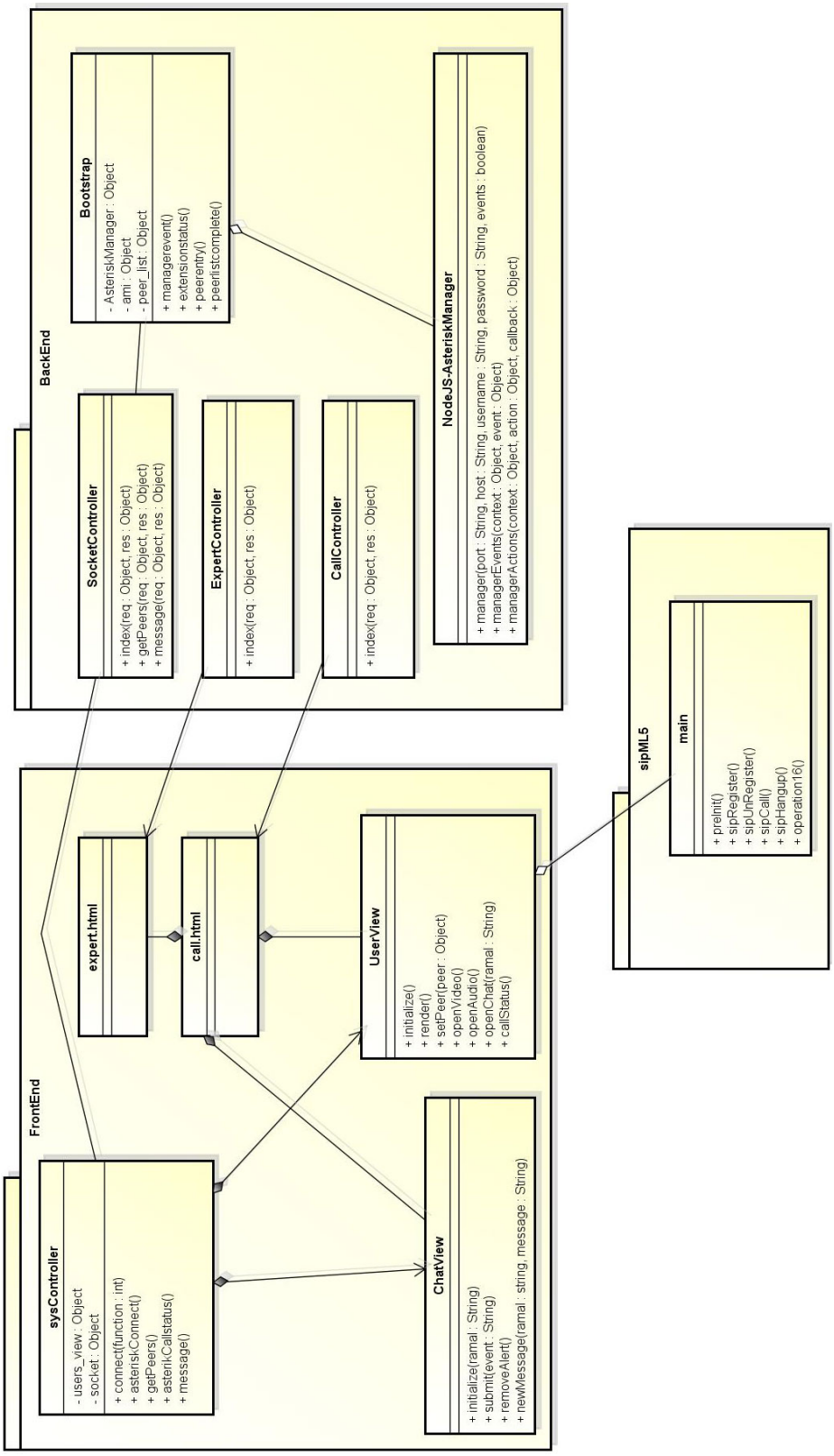

Figura 3.3: Diagrama de classe.

Fonte: Togo, 2015

CallController e ExpertController

Responsável por criar as rotas dos arquivos html, para ficarem acessíveis pelo navegador. O CallController fica responsável por apresentar a página principal da aplicação, enquanto o

ExpertController se encarrega de mostrar a tela de configurações avançadas aos usuários.

SysController

O SysController é a parte da aplicação que recebe todas as informações do *back-end*, e mostra ao usuário através da página do navegador. Todas as informações recebidas são tratadas e organizadas de forma a fazer com que o usuário possa usufruir de alguma maneira, seja para listar os usuários *online*, realizar ligações ou enviar mensagens instantâneas.

UserView

Parte da aplicação que possibilita que todos os ramais listados sejam alvos de alguma ação (chamada de vídeo, voz ou *chat*). Ou seja, adiciona algumas funcionalidades nos botões de cada ramal, juntamente com o número e o estado do ramal.

ChatView

Todo o envio e recebimento de mensagens via *chat* é tratado pelo ChatView. Em termos práticos, toda vez que algum usuário abrir uma janela de *chat* com algum outro ramal e enviar uma mensagem, o ChatView executa uma função informando para o SocketController sobre os participantes da conversa, e o conteúdo que está sendo enviado. Este encaminha via *socket* para o usuário destino, disparando a abertura de uma janela no navegador, contendo a mensagem.

3.4.2 Diagramas de sequência

A seguir, será mostrado alguns Diagramas de Sequência para melhor entendimento da aplicação e como elas estão interagindo entre sí.

A Figura 3.4 mostra um usuário acessando o sistema ao mesmo tempo em que está se conectando ao servidor Astersk, inserindo o número do seu ramal e senha que foram registrados previamente e endereço IP do servidor. O diagrama 3.5 é um exemplo de ligação entre dois ramais conectados no Asterisk. É basicamente uma ligação, atendimento e encerramento da ligação por parte do ramal destino. O diagrama 3.6, mostra o funcionamento da listagem dos ramais no sistema. Mostra desde a abertura de *socket* com o Asterisk, envio da *action*, coletando informações dos ramais, até o tratamento desta e impressão dos usuários ativos na tela. A Figura 3.7 mostra como é realizada a troca de mensagens entre dois usuários do sistema. São mostrados os dois caminhos: envio e recebimento das informações.

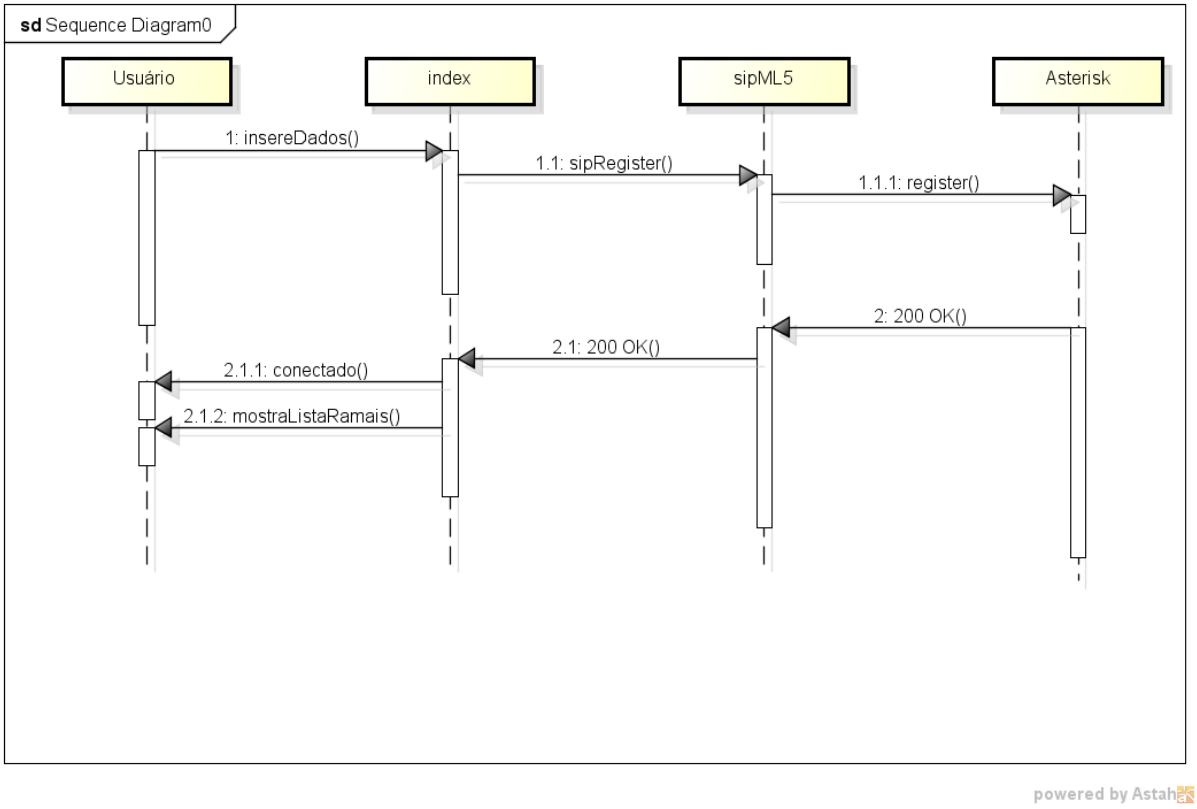

Figura 3.4: Diagrama de sequência - Register.

Fonte: Togo, 2015

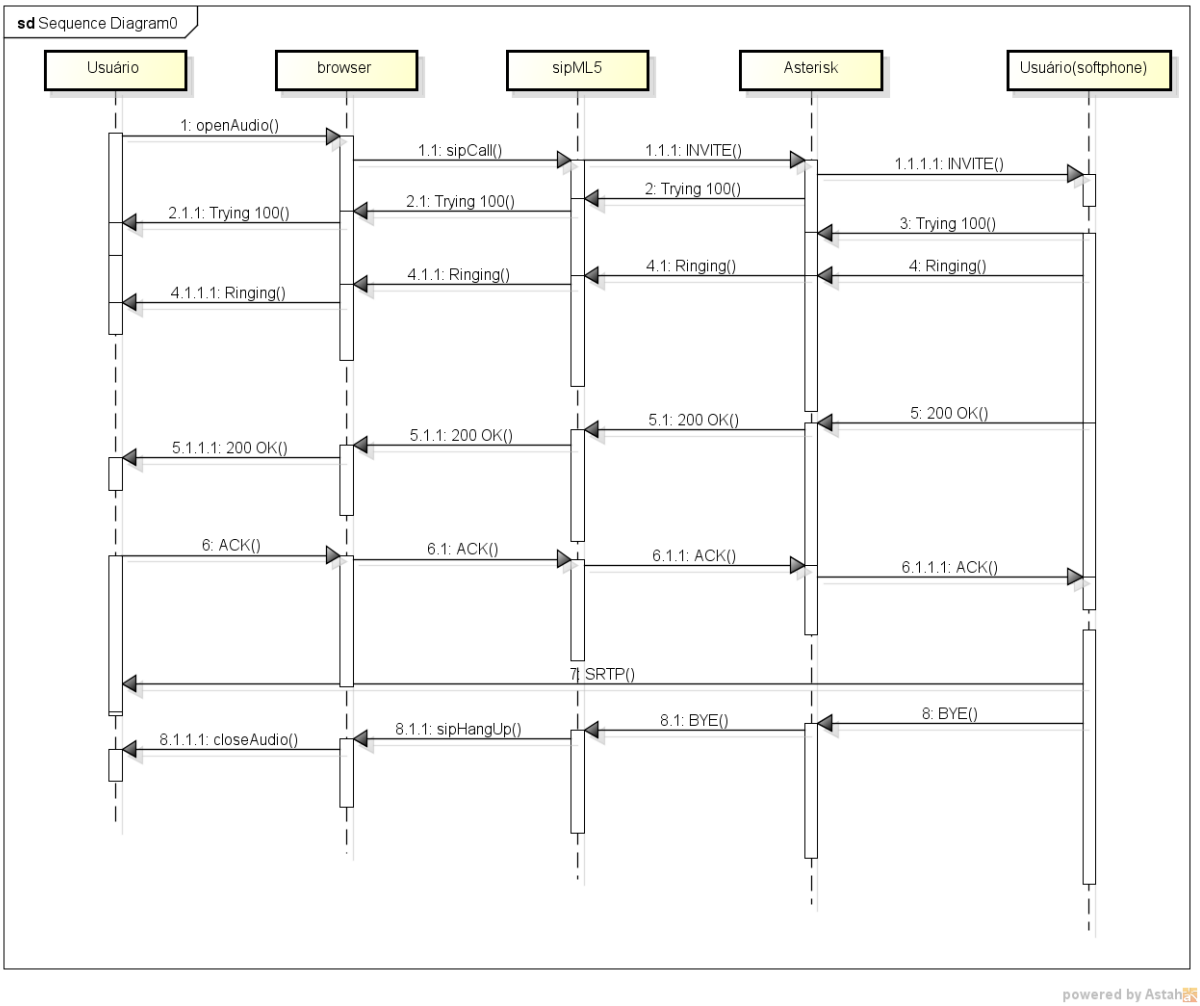

Figura 3.5: Diagrama de sequência - Invite.

Fonte: Togo, 2015

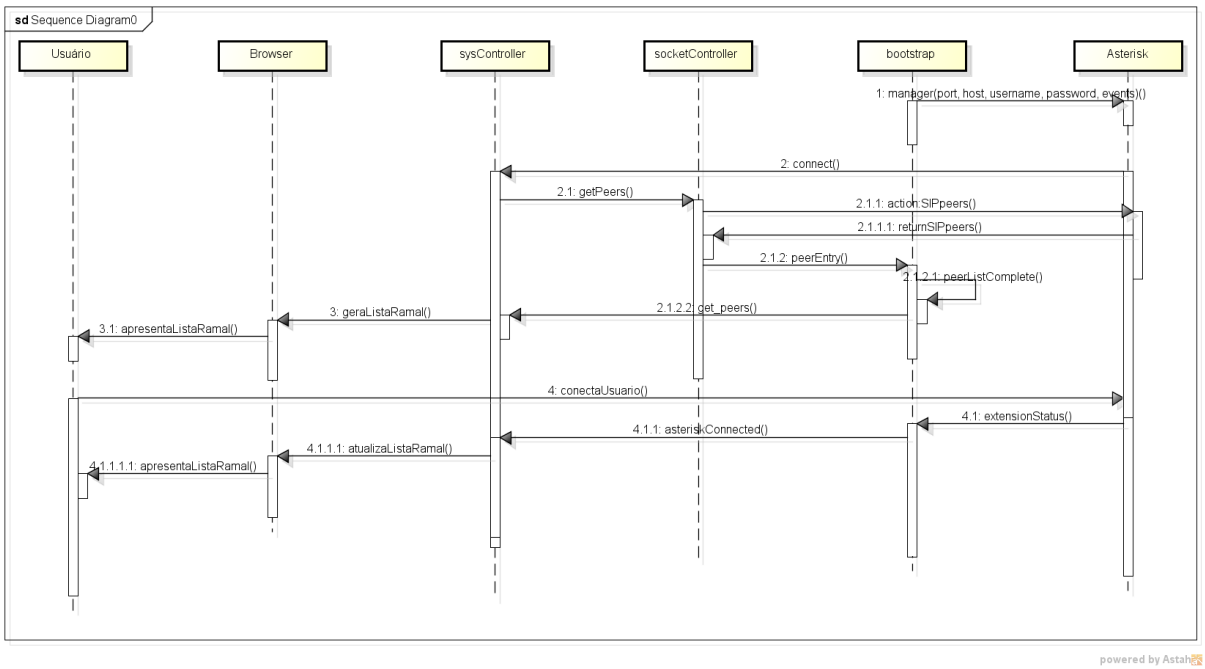

Figura 3.6: Diagrama de sequência - Lista de ramais.

Fonte: Togo, 2015

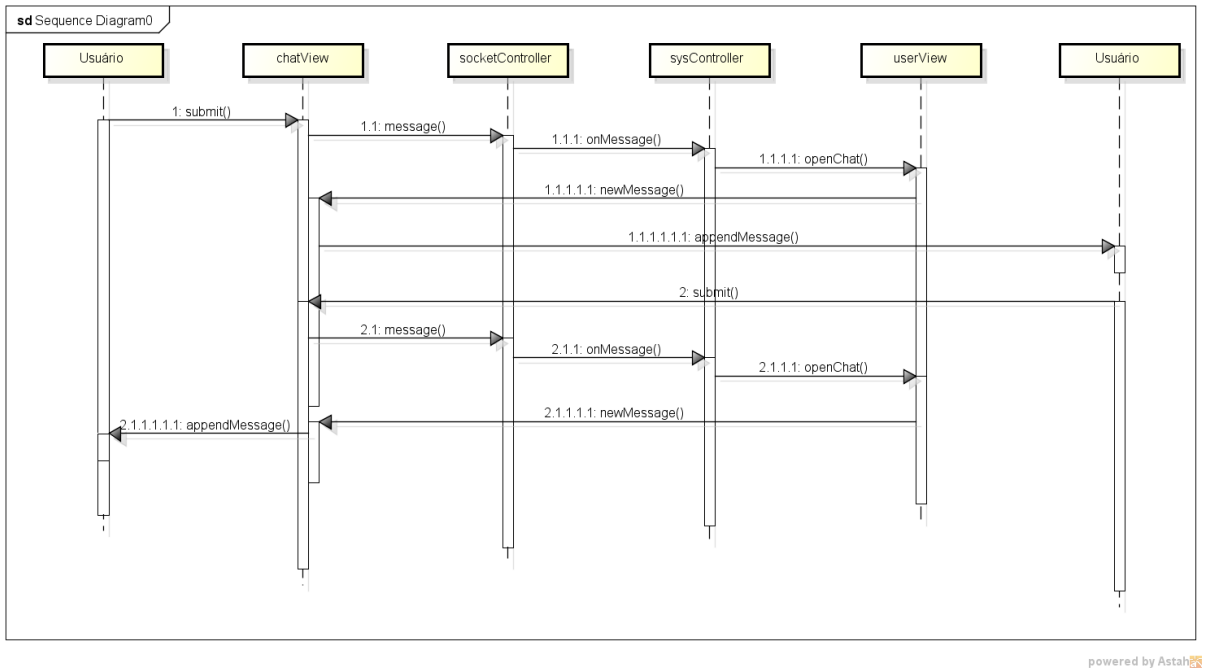

Figura 3.7: Diagrama de sequência - *chat*.

Fonte: Togo, 2015

4 *Implementação do sistema*

Este capítulo aborda a implementação da aplicação *web* utilizando a base do sipML5, juntamente com a configuração do Asterisk. Serão abordadas também as decisões do projeto, uso das principais bibliotecas empregadas, com destaque o Socket.io, a configuração do cenário de teste, dificuldades encontradas durante o decorrer do projeto e soluções para resolver os problemas encontrados.

4.1 Decisões do projeto

Como o objetivo do projeto é dar continuidade a um TCC desenvolvido anteriormente no IFSC-SJ, houve a necessidade de criar um cenário integrando o sipML5 com o servidor Asterisk. Além de criar o cenário e realizar testes para relatar o desempenho desta interação, foi decidido adaptar a ferramenta (sipML5), desenvolvendo novas funcionaliades para torná-la mais robusta.

No início deste projeto, havia pouco *software* implementando bem toda a pilha do protocolo SIP em Javascript. O único que se apresentou grande no cenário até então foi o sipML5. Este disponibilizava apenas o serviço de ligação áudio/vídeo de uma ponta a outra, sendo necessário primeiramente preencher alguns campos para o registro num servidor SIP. E foi sobre esta plataforma que toda a aplicação foi desenvolvida, e testada.

Posteriormente surgiu outra aplicação que desempenhava o mesmo papel do sipML5, o jsSIP[1]. Foram feitos testes supeficiais em seu site de demonstração. Primeiramente foram utilizados dois *softphones* comuns (xLite) conectado ao Asterisk, que funcionaram sem problemas no recebimento e envio de chamadas, havendo apenas um pequeno atraso no recebimento do áudio. Utilizando o jsSIP em uma ponta e o xLite em outra, houve alguns problemas tanto no Firefox quanto no Chrome. Ao realizar uma ligação, mostrava apenas uma mensagem de "chamando", porém não tocava no ramal destino. Dado o não funcionamento correto da ferramenta

[1] http://jssip.net/

e o fato de parte do projeto já estar implementada sobre o sipML5, foi decidido mantê-lo.

Em relação à central VoIP, foi decidido realizar os testes sobre o Asterisk, ao invés do FreeSwitch[2], por dois grandes motivos. Um deles é continuar o projeto do ex-aluno Felipe Borges, mantendo as mesmas tecnologias, relatando com mais detalhes os resultado obtidos. O outro motivo foi o fato do Asterisk possuir uma melhor documentação sobre a interação com o WebRTC.

A implementação do estado do ramal, é possível fazer de maneira nativa do próprio SIP, porém foi decidido fazer diretamente com Javascript, para abstrair problemas aos desenvolvedores *web*. Não há necessidade de entender todo o protocolo SIP, para desenvolver a mesma funcionalidade. O intuito é mostrar que o WebRTC disponibilizou uma nova forma de implementar sistemas de comunicação em tempo real, utilizando linguagens já conhecidas no ambiente *web*.

4.2 Desenvolvimento das funcionalidades

Nos tópicos a seguir, será descrita o desenvolvimento de toda a aplicação, algumas das decisões tomadas durante o projeto e como é o relacionamento entre as funcionalidades.

4.2.1 Sistema de acesso

Como foi mencionado anteriormente, foi utilizado a base do sipML5 para o desenvolvimento de uma nova aplicação *web*. Foram realizadas algumas alterações do sistema original, pois além de criar novas funcionalidades, umas das propostas deste projeto era o de criar uma interface intuitiva, para que o usuário ao acessar, saiba como interagir com outros usuários conectados.

Ao acessar a página da aplicação, o usuário precisa preencher alguns campos para se conectar e liberar as funcionalidades disponíveis no sistema, como mostra a Figura 4.1. É importante lembrar que o usuário e a senha, foram previamente gerados no arquivo `sip.conf` do Asterisk onde será mostrado mais adiante. Após preencher todos os campos, é realizado a validação do usuário. Caso o usuário e a senha estiverem corretas, é retornado um 200 OK do Asterisk, fazendo com que a tela de bloqueio desapareça, liberando acesso pleno as funcionalidades do sistema. Veja nas Figuras 4.2 4.3 respectivamente.

Paralelo a isto, é criado pelo sipML5 um objeto contendo um conjunto de informações necessárias para realizar qualquer outra ação (chamadas de voz, vídeo) dentro do sistema. Observe

[2]https://freeswitch.org/

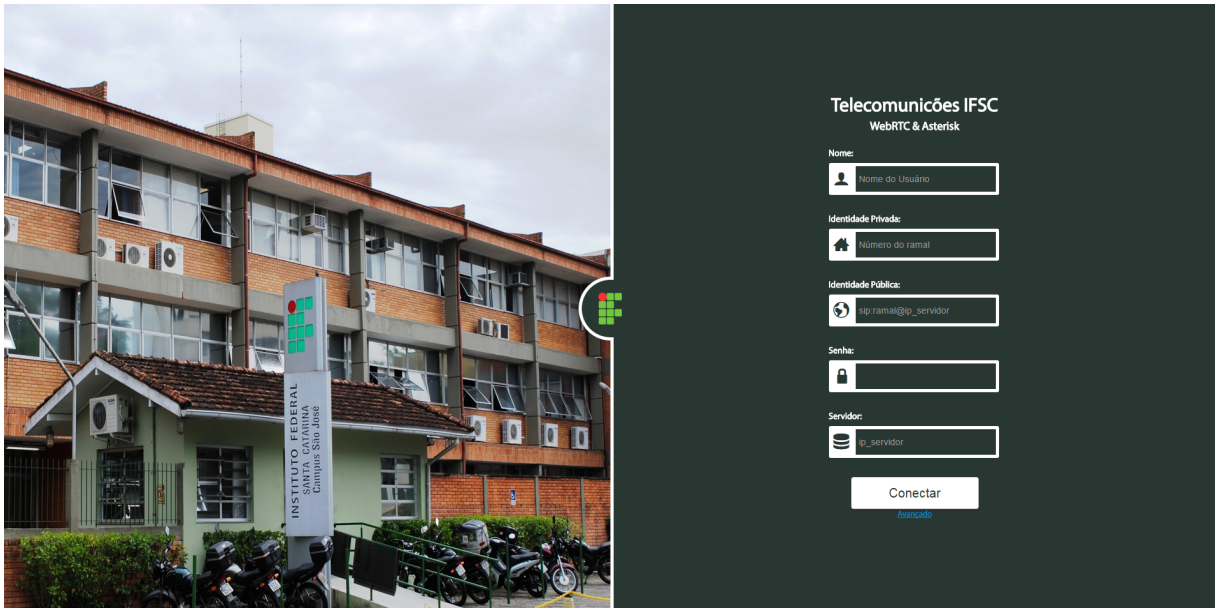

Figura 4.1: Tela de login.

Fonte: Togo, 2015

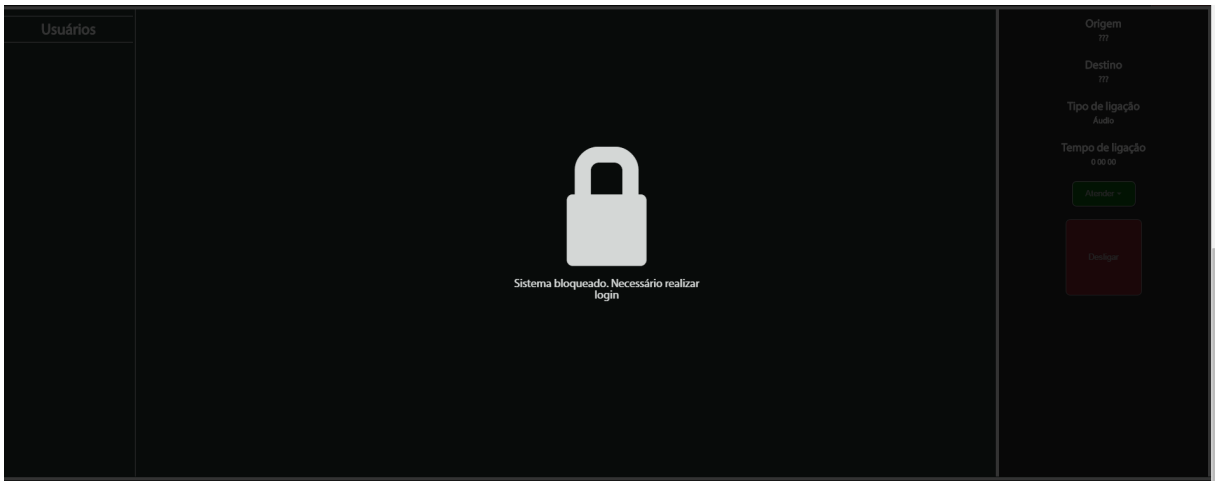

Figura 4.2: Tela de bloqueio. Liberado após usuário acessar o sistema.

Fonte: Togo, 2015

a Figura 4.4.

4.2.2 Lista de contatos e estado de presença

Após liberado o acesso ao sistema, é disponibilizado ao usuário a lista de ramais registrados no Asterisk. Ao lado de cada ramal, foram criados algumas funções possibilitando uma chamada de vídeo, áudio ou/e *chat*. Figura 4.5.

Para gerar esta lista, foi utilizada a API NodeJS Asterisk Manager onde é aberta um *socket* com a AMI do Asterisk, possibilitando o envio de comandos (*actions*) e/ou escuta de eventos

Figura 4.3: Tela principal da aplicação.

Fonte: Togo, 2015

```
var o_stack = new SIPml.Stack({
        realm: 'example.org',
        impi: 'bob',
        impu: 'sip:bob@example.org',
        password: 'mysecret', // optional
        display_name: 'I Am Legend', // optional
        websocket_proxy_url: 'ws://192.168.0.10:5060', // optional
        outbound_proxy_url: 'udp://192.168.0.12:5060', // optional
        ice_servers: [{ url: 'stun:stun.l.google.com:19302'}, { url:'turn:user@numb.viagenie.ca', credential:'myPassword'}],
        bandwidth: { audio:64, video:512 }, // optional
        video_size: { minWidth:640, minHeight:480, maxWidth:1920, maxHeight:1080 }, // optional
        enable_rtcweb_breaker: true, // optional
        enable_click2call: false, // optional
        events_listener: { events: '*', listener: listenerFunc }, //optional
        sip_headers: [ //optional
            {name: 'User-Agent', value: 'IM-client/OMA1.0 sipML5-v1.0.89.0'},
            {name: 'Organization', value: 'Doubango Telecom'}
        ]
    }
);
```

Figura 4.4: Objeto criado, após registro do ramal.

Fonte: Togo, 2015

em tempo real, retornando a aplicação informações que podem ser tratadas e utilizadas. Para tornar isso possível, foi necessário configurar no Asterisk o arquivo `manager.conf`. Além da alteraçção de alguns campos, foi adicionado um usuário, senha e permissões de leitura e escrita, para que este tenha acesso externo. Na Figura 4.6 mostra com detalhes a configuração deste usuário.

As atribuições feitas abaixo do campo *general* (linhas 5 à 9) são consideradas globais, ou seja, todos os outros usuários criados neste mesmo arquivo, herdarão estes atributos, como é o caso do usuário admin. A linha 6 habilita a AMI do Asterisk, linha 7 permite acesso a AMI via HTTP, linha 8 por padrão vem configurado a porta 5038 para as conexões AMI e a linha 10

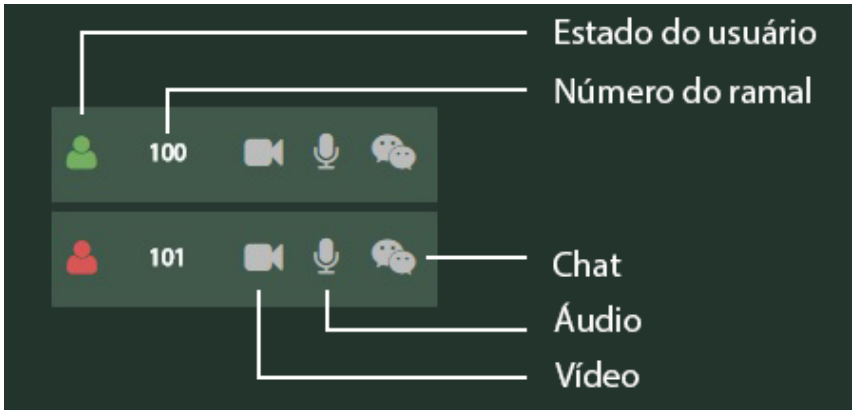

Figura 4.5: Usuários gerados dinâmicamente.

Fonte: Togo, 2015

```
3
4   [general]
5   displaysystemname = yes
6   enabled = yes
7   webenabled = yes
8   port = 5038
9   httptimeout = 300
10  bindaddr = 0.0.0.0
11
12
13  [admin]
14  secret = 1234
15  read = system,call,log,verbose,agent,user,config,command,dtmf,reporting,cdr,dialplan,originate
16  write = system,call,log,verbose,agent,user,config,command,dtmf,reporting,cdr,dialplan,originate
17  permit=0.0.0.0/0.0.0.0
18
```

Figura 4.6: Configuração do arquivo manager.conf.

Fonte: Togo, 2015

define que qualquer endereço pode escutar as conexões com a AMI.

Para configuração individual de cada usuário, foi criado o usuário admin. Nele foi definido uma senha (linha 14), tudo o que o usuário pode executar ou receber do Asterisk (linhas 15 e 16) e qual endereço IP pode se conectar com a AMI com o usuário especificado (linha 17).

Após a configuração do usuário, ao rodar a aplicaçao *web*, esta deve abrir uma conexão com o Asterisk para envio e recebimento de dados. Utilizando a API mencionada anteriormente, é criado um objeto que será responsável por esta conexão com a central. Nele deve ser informada a porta de acesso, endereço IP onde se encontra a central, o nome e a senha do usuário que foi criado no Asterisk e se os eventos serão emitidos. Veja um exemplo na Figura 4.7. A linha 26 faz a chamada da API enquanto a linha 27 faz a criação do objeto. A linha 29 é resposável por manter a conexão com a AMI, linhas 31 à 34 são responsáveis por ficar ouvindo alguns eventos de exemplo, onde foram declarados como primeiro parâmetro (*managerevent*, *hangup*, *confbridgejoin*, *response*), e as linhas 36 à 46 são responsáveis por executar um comando no

Asterisk.

```
25
26  var AsteriskManager = require('asterisk-manager'),
27      ami = new AsteriskManager('5038','189.114.205.190','admin','1234', true);
28
29  ami.keepConnected();
30
31  ami.on('managerevent', function(evt) {});
32  ami.on('hangup', function(evt) {});
33  ami.on('confbridgejoin', function(evt) {});
34  ami.on('response', function(evt) {});
35
36  ami.action({
37    'action':'originate',
38    'channel':'SIP/myphone',
39    'context':'default',
40    'exten':1234,
41    'priority':1,
42    'variable':{
43      'name1':'value1',
44      'name2':'value2'
45    }
46  }, function(err, res) {});
47
```

Figura 4.7: Abrindo conexão com a AMI do Asterisk. Escuta de eventos e envio de *actions*.

Fonte: Togo, 2015

Para a captura dos ramais registrados, é enviada uma *action* chamada SIPpeers, que retornará alguns eventos do tipo `peerEntry`, contendo o número de cada ramal, número da porta, endereço IP e outras informações. Logo após, o Asterisk retorna outro evento chamado `peerListComplete`, informando a quantidade de ramais registrados. Com esta informação, foi criado uma função que gera uma lista de usuário de maneira dinâmica, ou seja, se forem adicionados outros ramais no Asterisk, a aplicação mostrará em tempo real. Na Figura 4.8 mostra as informações que o evento peerEntry retorna.

Em relação ao estado de presença dos ramais, toda vez que um usuário se conectar ao Asterisk, é disparado um evento chamado extensionStatus, onde a aplicação irá coletar e tratar, de forma que mostre aos demais usuários, que alguém se conectou na central.

Na Figura 4.9, mostra todo o processo desde a geração da lista de usuários, até a alteração do estado de presença de um ramal, após a sua entrada.

A lista de eventos e ações estão disponibilizadas na documentação[3] do Asterisk. A cada versão o número de eventos e ações vem crescendo, possibilitando um maior número de customização do sistema. Alguns exemplos disponíveis no Apêndice C.

[3]https://wiki.asterisk.org/wiki/display/AST/Asterisk+13+Documentation

```
managerevent { event: 'PeerEntry',
  actionid: '1435283270792',
  channeltype: 'SIP',
  objectname: '101',
  chanobjecttype: 'peer',
  ipaddress: '192.168.1.11',
  ipport: '38241',
  dynamic: 'yes',
  autoforcerport: 'no',
  forcerport: 'no',
  autocomedia: 'no',
  comedia: 'no',
  videosupport: 'no',
  textsupport: 'no',
  acl: 'no',
  status: 'Unmonitored',
  realtimedevice: 'no',
  description: '' }
managerevent { event: 'PeerlistComplete',
  actionid: '1435284661666',
  eventlist: 'Complete',
  listitems: '11' }
```

Figura 4.8: Eventos retornados após o envio da *action* SIPpeers

Fonte: Togo, 2015

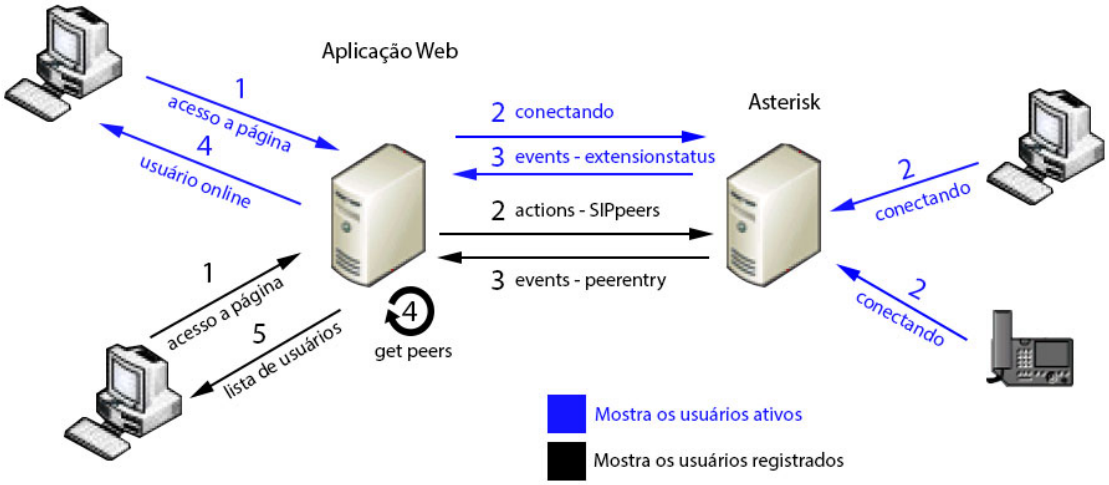

Figura 4.9: Visão geral da lista de contato de estado de presença dos ramais

Fonte: Togo, 2015

4.2.3 Mensagens Instântaneas

Toda a estrutura do *chat*, foi construida através de um módulo do Node.js chamado Socket.io, explicado na seção 2.6. Sua utilização, se dá no clique no botão de *chat* de um usuário da lista, onde abrirá uma janela para o envio e recebimento de mensagens. Este envio é realizado pela função chamada *submit*, que é disparada através de um evento de clique no botão. A Figura 4.10 apresenta um exemplo de conversação entre dois ramais conectados na central. Vale ressaltar que as mensagens transitam apenas na parte da aplicação, não chegando a passar nada ao Asterisk.

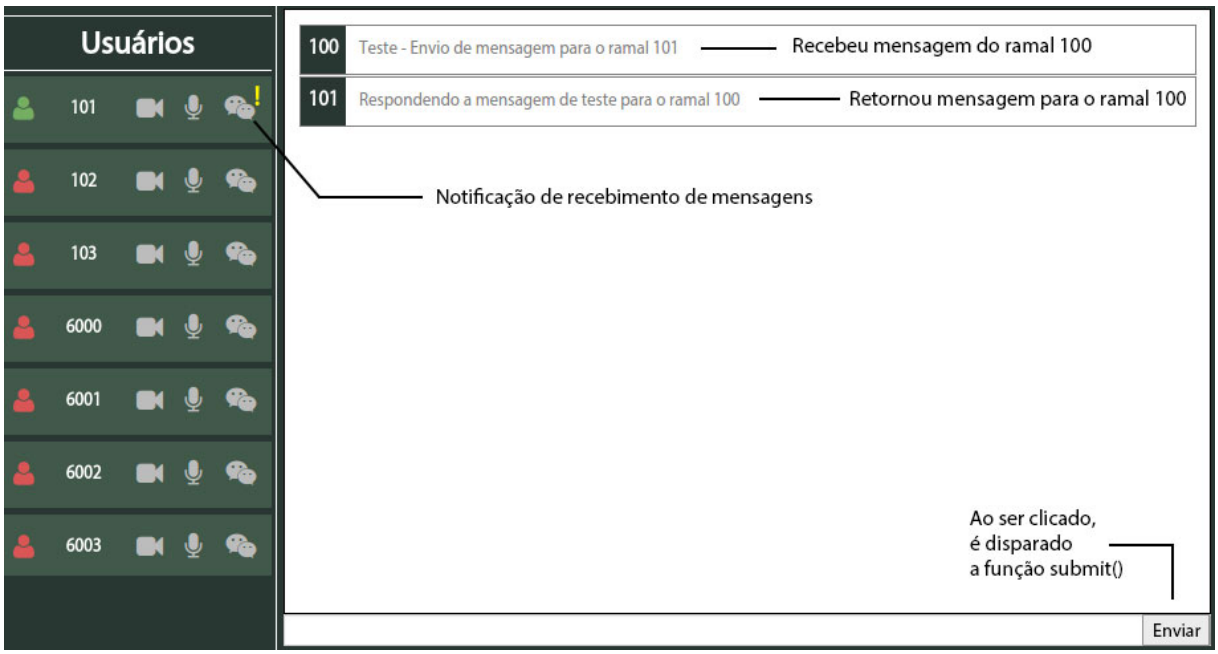

Figura 4.10: Exemplo de conversação via *chat*

Fonte: Togo, 2015

Após o clique para o envio da mensagem, a função *submit* é disparada capturando algumas informações que foram inseridas ao se conectar no sistema. Estas informações são enviadas para o *back-end* da aplicação, que tem como tarefa encaminhar para o outro usuário. A mensagem retorna novamente para o *front-end*, disparando uma nova janela ao usuário destino, contendo o conteúdo da mensagem. Este, ao responder, refaz todas as etapas anteriormente mencionadas. Toda a troca de mensagem encaminha apenas três informações: ramal de origem, ramal de destino e conteúdo da mensagem transmitida. A partir disso a aplicação fica responsável por mostrar a mensagem para o usuário. Vale ressaltar, que a mensagem fica disponível apenas para usuários ativos, e estas não são armazenadas, deixando de existir caso usuário relogar no sistema. Na Figura 4.11 há uma visão geral das etapas de conversação entre dois usuários.

4.2.4 Estabelecimento de chamadas

Para o estabelecimento de chamadas, a parte funcional do sipML5 continou a mesma, sendo necessário apenas de algumas alterações na parte visual, para melhorar a usabilidade do usuário com o sistema. Foram dispostos de maneira simplificada as opções de chamada áudio, vídeo e envio de mensagens instantâneas, conforme a Figura 4.4, não sendo necessário digitar manualmente o número do ramal em que deseja se comunicar. Vale ressaltar que o sistema tinha como premissa criar um ambiente privado onde somente quem estivesse conectado com a cen-

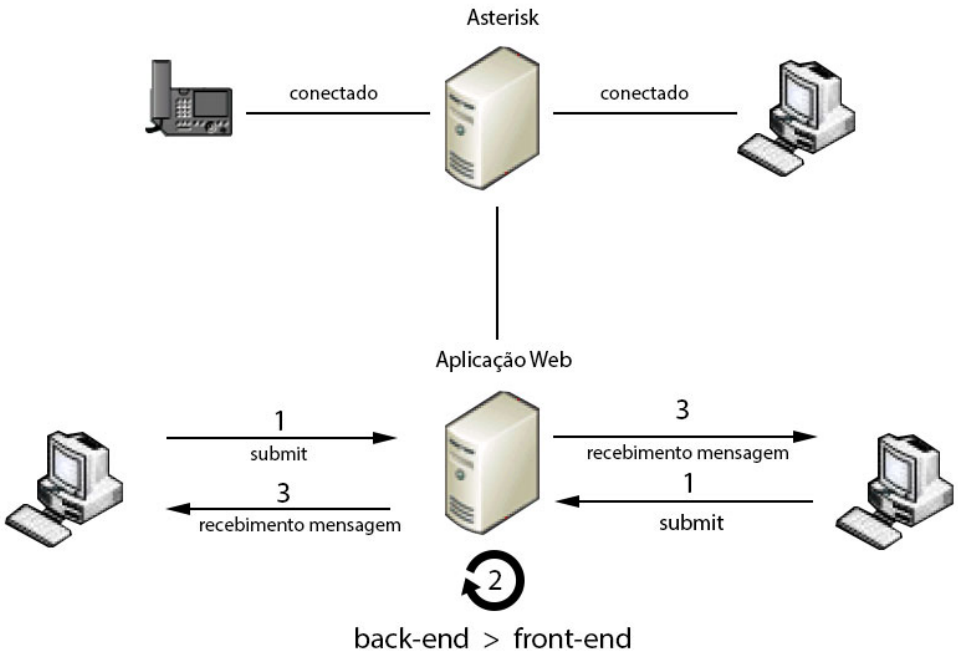

Figura 4.11: Conversação entre dois usuários via mensagem instantânea.

Fonte: Togo, 2015

tral pudesse se comunicar, impossibilitando ligações para PSTN[4]. Abaixo na Figura 4.12, o site padrão do sipML5.

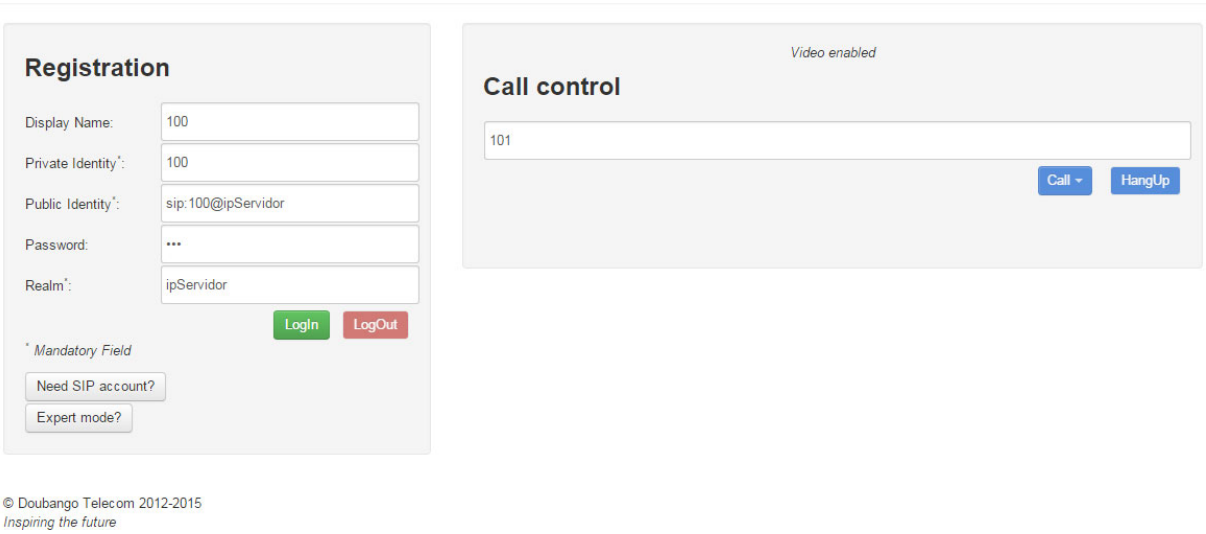

Figura 4.12: *Layout* padrão do sistema sipML5.

Fonte: sipml5.org, 2012

Ao realizar uma ligação, o objeto que foi criado ao se conectar no sistema (Figura 4.4) é

[4]Rede Pública de Telefonia Comutada. Termo usado para identificar a rede telefônica mundial comutada por circuitos destinada ao serviço telefônico, sendo administrada pelas operadoras de serviço telefônico.

utilizado para montar a pilha SIP e enviar a sinalização para o ramal destino a fim de criar um canal de comunicação. Primeiramente é criado uma nova sessão do tipo *call-audiovideo*, para posteriormente utilizar o método *call*, que de fato fará a ligação. Neste método é necessário declarar o nome do destino, número do ramal ou identificador (sip:usuario@servidor.com' ou 'usuario' ou '100') e algumas informações de configuração. Observe na Figura 4.13.

Figura 4.13: Método responsável por realizar uma chamada entre dois ramais

Fonte: Togo, 2015

Realizando ou recebendo uma ligação, é capturada do Asterisk um evento chamado *bridge*, que contém informações relevantes sobre a chamada. Com estas informações, foi possível criar um painel que informa quem está em conversação no momento e quando esta foi inicializada. Ao finalizar a chamada, estes eventos são disparados novamente e a função da aplicação consegue imprimir ao usuário o tempo total da ligação. Figura 4.14

4.3 Dificuldades encontradas e suas soluções

Durante o desenvolvimento da aplicação, houve algumas dúvidas e problemas que acabaram impactando no andamento do projeto. Foram tomadas medidas corretivas e mudanças em partes do projeto.

4.3.1 Geração da lista de usuários via PHP

Para o desenvolvimento da lista de usuários, foi utilizado inicialmente a linguagem PHP para a captura de eventos e/ou envio das *actions* ao Asterisk. Houve sucesso na captura das informações, porém houve grandes dificuldades ao tratar estas informações e disponibilizar aos clientes. A parte do código responsável por essa tarefa se mostrou muito ineficiente, pois era necessário enviar periodicamente requisições ao Asterisk para receber o estado dos ramais em tempo real, e toda vez que isso ocorria, havia a atualização da página, interrompendo o uso contínuo da ferramenta.

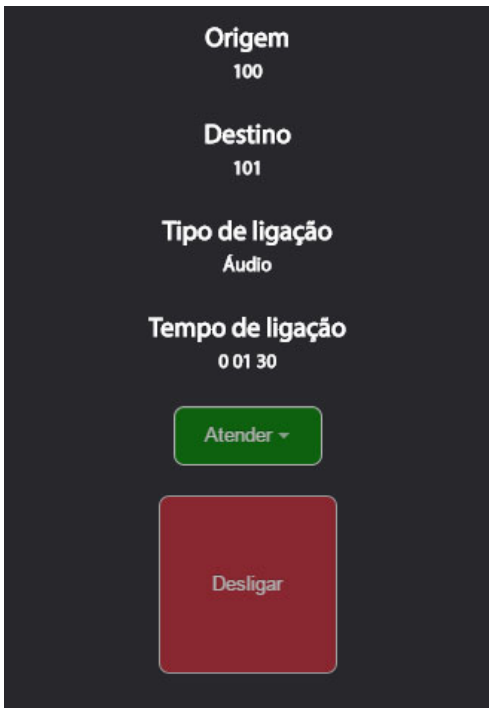

Figura 4.14: Painel informando sobre os participantes e o tempo da ligação

Fonte: Togo, 2015

Pela falta de domínio da linguagem, ocasionando um baixo desempenho no sistema, viu-se a necessidade de mudança no projeto. Após algumas pesquisas, o Javascript apareceu como a melhor solução para o cenário. Ao invés de gerar diversas requisições ao Asterisk, é aberto um *socket* com o mesmo, que ficará ouvindo todos os eventos executados. Ou seja, caso um usuário se conectar no sistema, a aplicação receberá do Asterisk este evento, fará o tratamento e enviará para a interface da aplicação, informando que o usuário está ativo.

4.3.2 Integração Sails.js e sipML5

Uma das grandes dificuldades enfrentadas foi integrar o projeto do sipML5 já customizada com a nova interface, com o *framework* Sails.js que entrega um padrão de projeto organizado e que já incorpora o Socket.io em sua utilização. Foi feito desta maneira, pois houve dúvidas de como seria a melhor maneira de organizar todo o projeto, evitando sair das boas práticas.

Como o projeto do sipML5 é relativamente grande, ao fazer a integração ocorreram diversos erros acusando falta de arquivos, erro de partes do código, e outros erros para os quais não foi possível encontrar auxílio na Internet. Após diversos testes, todos os erros foram solucionados e toda a aplicação rodou sem problemas. Apesar de solucionados, há necessidade também do apronfundamento do *framework*, para uma melhor estruturação.

Sendo o Sails um *framework* construído em Node.js, o uso do módulo Socket.io se tornou prático para a construção do serviço de *chat* e para a listagem dos ramais registrados no Asterisk.

4.4 Fechamento

A parte de desenvolvimento das novas funcionalidades foram bastante desafiadoras, pois foram utilizadas tecnologias que podem ser consideradas recentes em comparação com as demais linguagens utilizadas hoje no mercado. O uso de um dos módulos do Noje.js, o Socket.io, foi possível fazer de maneira relativamente simples, toda a listagem dos ramais registrados no Asterisk, alteração dos estados dos ramais em tempo real e implementação do *chat* da aplicação.

O fato da API(Socket.io) ter auxiliado o desenvolvimento, não está relacionado com o grau de dificuldade no desenvolvimento do projeto. Foi necessário estudar casos de usos, ler e compreender a documentação desta API, para entender quais funcionalidades e como estes eram utilizados, realizar testes nos casos onde apresentado algum problema de implementação.

É bastante comentado na comunidade *web*, que o Node.js, juntamente com seus módulos, irão revolucionar a maneira de como utilizamos a Internet hoje. Grandes aplicações poderão rodar diretamente na nuvem, possuindo desempenhos equivalentes com programas instalados em computadores hoje.

5 *Testes e resultados obtidos*

Após e durante a implementação do sistema, foi realizado uma bateria de testes para encontrar soluções e identificar possíveis *bugs* que poderiam ocorrer após o uso da aplicação. Este capítulo apresentará como estes testes foram realizados e quais foram os resultados obtidos.

5.1 Ambiente de testes

Todos os testes foram efetuados em máquinas virtualizadas dentro da infraestrutura do OpenStack[1] do campus IFSC-SJ. Foi gerado uma máquina com o IP 200.135.233.54, utilizando o sistema operacional Ubuntu 14.04.2 LTS com o Asterisk 11.18.0. A instalação completa do cenário é relatada no Apêndice A. Na Figura 5.1 representa o cenário real em que houve os testes.

5.2 Testes realizados

Foi realizada uma bateria de testes para descobrir se todas as funcionalidades criadas na aplicação estavam funcionando de maneira correta e sem possíveis *bugs*, e se toda a interação do sipML5 com o Asterisk foi realizada com êxito.

5.2.1 Integraçao do sipML5 com o Asterisk/teste de ligações

Uma das maneiras de saber se houve sucesso na interação, é realizando testes de ligações utilizando as duas tecnologias (sipML5 e Asterisk).

Como mencionado no início do tópico, os testes foram realizados utilizando como servidor uma máquina virtual situada no campus IFSC-SJ. Foram utilizados computadores de mais de

[1] OpenStack é um *software* de código aberto, capaz de gerenciar os componentes de múltiplas infraestruturas virtualizadas, assim como o sistema operacional gerencia os componentes de nossos computadores, o OpenStack é chamado de Sistema Operacional da Nuvem, por cumprir o mesmo papel em maior escala.

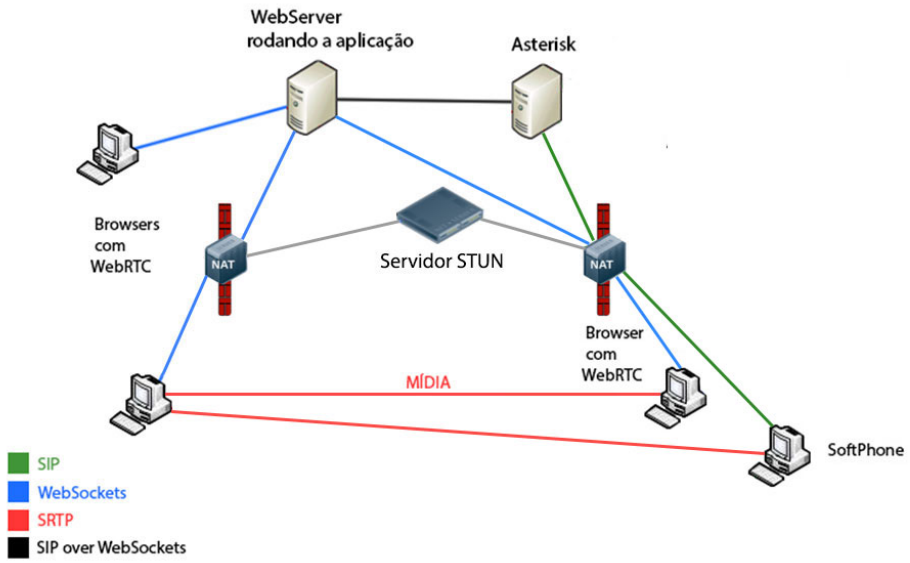

Figura 5.1: Cenário de testes

Fonte: Togo, 2015

uma localidade para checar o funcionamento do serviço.

Primeiramente foi feito uma ligação entre um *softphone* comum e a aplicação desenvolvida utilizando a base do sipML5. Foi utilizado o *softphone* X-Lite versão 4.8.4 76589 em uma ponta, e na outra foi utilizado o navegador Google Chrome 43.0.2357.134 (64bits). Vale lembrar que no navegador Chrome foi adicionado um certificado de segurança, pois no próprio site do Asterisk é ressaltado que nas ultimas versões do Google Chrome e Mozilla Firefox o uso do DTLS-SRTP é obrigatório.

Nas Figuras 5.2 é mostrado o certificado inserido no navegador, seguido das configurações utilizadas nos dois ramais de testes (Figura 5.3 e 5.4, 5.5).

A ligação apresentou boa qualidade, houve um pouco de atraso, provavelmente por causa do servidor, porém nada que pudesse atrapalhar a ligação. No console do Asterisk, com o comando `rtp set debug on` foi possível monitorar a troca de RTP entre os dois ramais, provando que a configuração utilizando o protocolo ICE para travessia de NAT, foi realizada com sucesso. Confira na Figura 5.6

Posteriormente foi realizado outro teste utilizando dois navegadores Chrome. Foram utilizados as mesmas configurações da Figura 5.4 e 5.5.

Realizado ligações de ambos os lados com sucesso. Houve um pequeno atraso na voz transmitida, porém nada que pudesse comprometer o funcionamento do sistema. Uma obsevação

Figura 5.2: Certificado adicionado no Google Chrome

Fonte: Togo, 2015

importante a ser levado em conta, é que no console do próprio navegador, há diversas informações sobre a ligação, e nela também é possível constatar o uso do protocolo ICE. Além de mostrar qual servidor STUN está sendo utilizado, são mostrados também dois atributos importantes para o uso do protocolo ICE: os atributos *ice-pwd* e *ice-ufrag* são utilizados para prover uma senha para proteger a conectividade STUN e prover fragmentos usados para construir o *username* da conectividade STUN. Veja na Figura 5.7

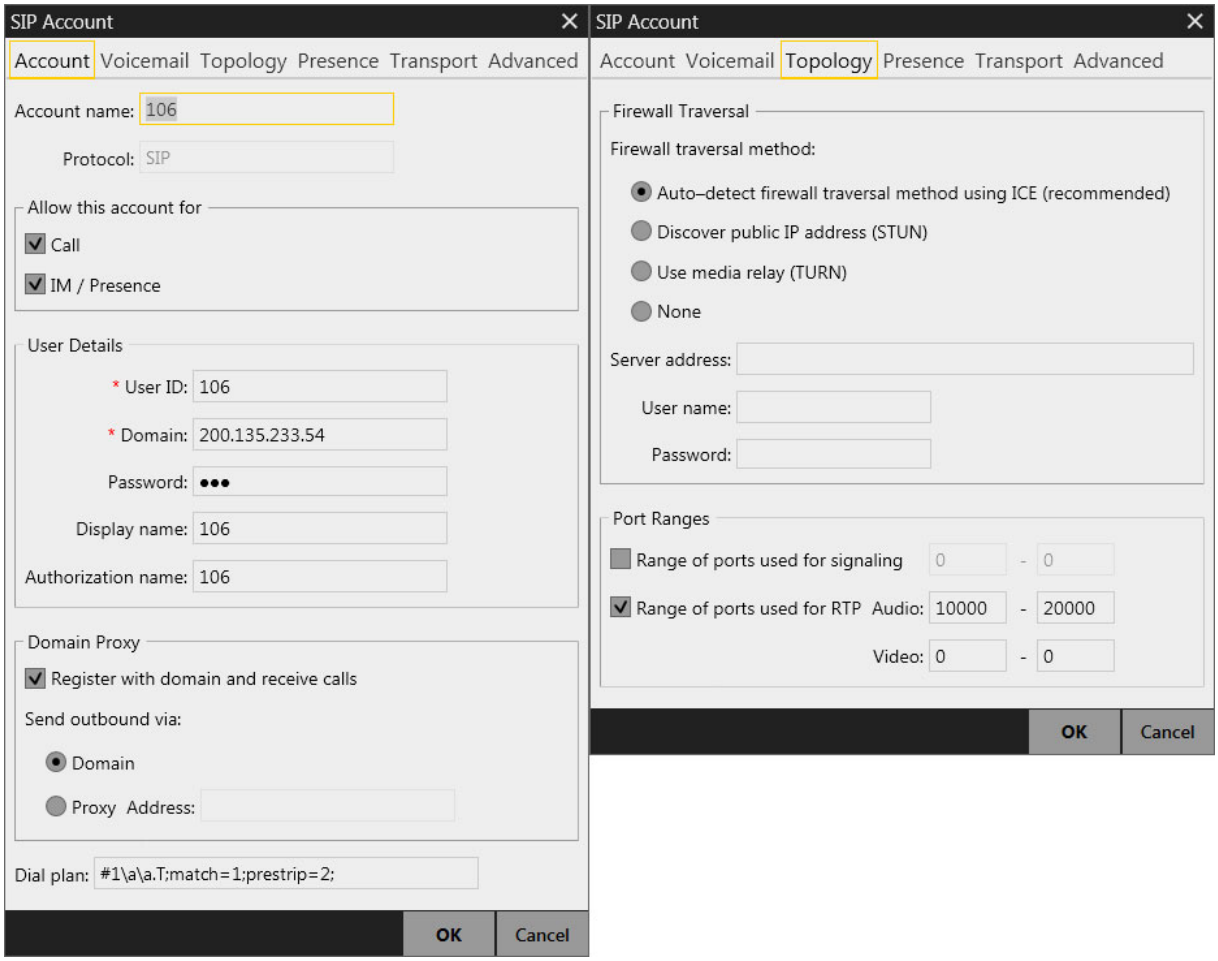

Figura 5.3: Configuração utilizado no X-Lite

Fonte: Togo, 2015

5.2.2 Lista de usuários e ramais conectados

Não houve grandes complicações para testar a lista de ramais registrados com os que estavam definitivamente ativos no momento. Para checar a lista de ramais, foi necessário adicionar novos ramais no arquivo de configuração do Asterisk, `contas-sip.conf` (onde é referenciado dentro do `sip.conf`) e `extensions.conf`. Após executar o comando de *reload* dentro do console do Asterisk, foi preciso verificar se, na aplicação, a lista de ramais tinha sido alterada. Todos os testes foram positivos.

Em relação aos testes dos estados dos ramais, foi necessário que alguns ramais se registrassem no Asterisk. Após a conexão ou desconexão do ramal com o Asterisk, foi preciso verificar primeiramente no console do Asterisk se a requisição chegou corretamente, para depois conferir na tela da aplicação. Testes e resultados realizados com sucesso.

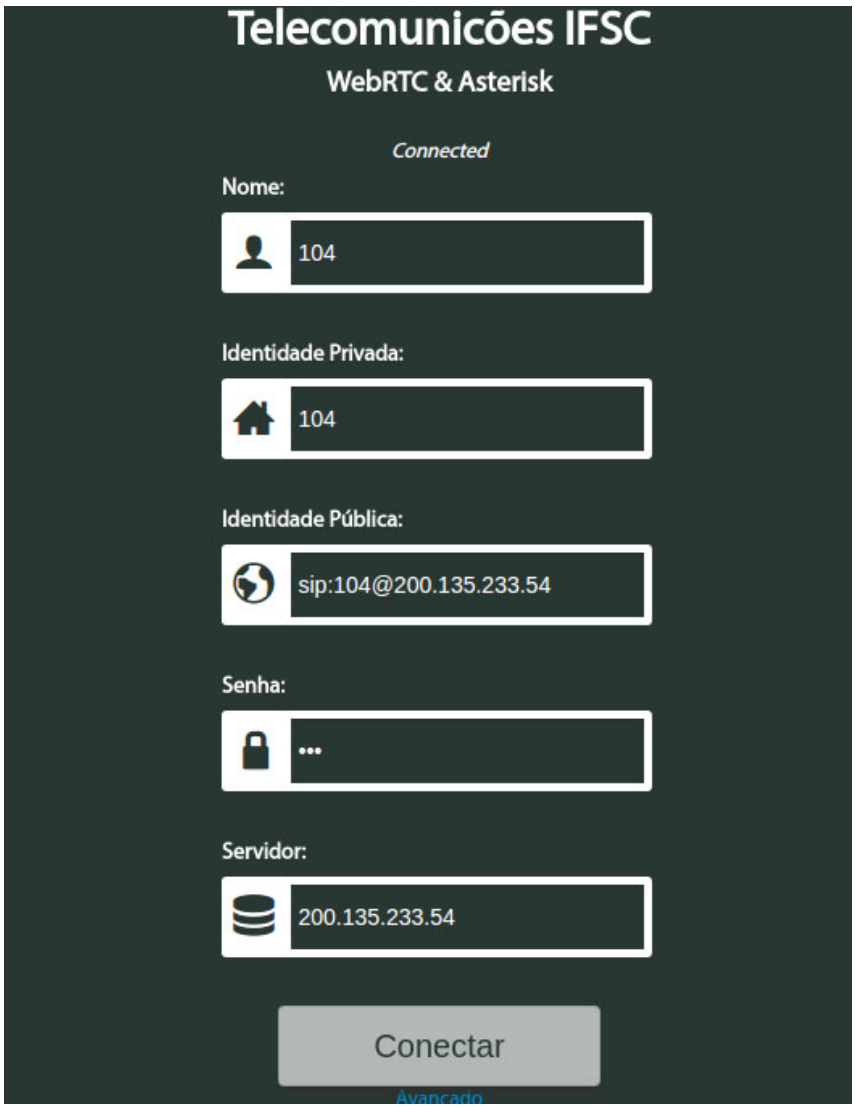

Figura 5.4: Configuração de login da aplicaçao desenvolvida

Fonte: Togo, 2015

5.2.3 *Chat*

Os testes de *chat* foram realizados com até quatro usuários conectados simultâneamente no sistema. Todos os ramais tiveram que enviar mensangens entre sí. Não houve nenhum tipo de problema de funcionamento e desempenho.

5.3 Conclusões

Durante toda a fase de desenvolvimento e interação, foram executados diversos testes logo após a implementação de novas partes do projeto, para evitar algum problema posterior que já pudesse estar em uma escala maior, dificultando a depuração do código.

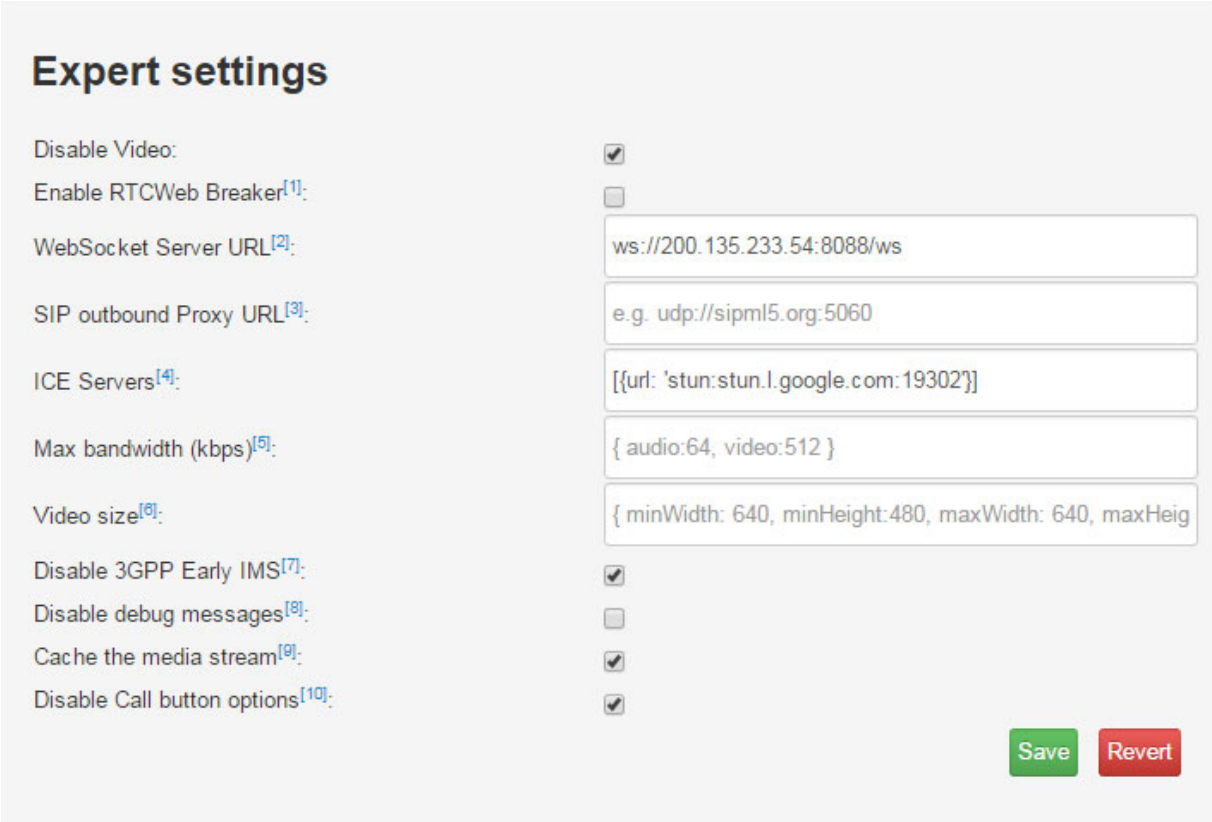

Figura 5.5: Configuração utilizada na configurações avançadas da aplicaçao desenvolvida

Fonte: Togo, 2015

```
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028820, ts 941725116, len 000160)
Sent RTP packet to      187.112.88.89:35543 (via ICE) (type 00, seq 017497, ts 941725112, len 000160)
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028821, ts 941725276, len 000160)
Sent RTP packet to      187.112.88.89:35543 (via ICE) (type 00, seq 017498, ts 941725272, len 000160)
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028822, ts 941725436, len 000160)
Sent RTP packet to      187.112.88.89:35543 (via ICE) (type 00, seq 017499, ts 941725432, len 000160)
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028823, ts 941725596, len 000160)
Sent RTP packet to      187.112.88.89:35543 (via ICE) (type 00, seq 017500, ts 941725592, len 000160)
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028824, ts 941725756, len 000160)
Sent RTP packet to      187.112.88.89:35543 (via ICE) (type 00, seq 017501, ts 941725752, len 000160)
Got  RTP packet from    187.112.88.89:11840 (type 00, seq 028825, ts 941725916, len 000160)
```

Figura 5.6: Configuração utilizado na aplicaçao desenvolvida

Fonte: Togo, 2015

Problemas em testes das funcionalidades em desenvolvimento já eram esperados como em qualquer outro desenvolvimento, porém nos testes relacionados à interação do sipML5 com o Asterisk ocorreram diversos problemas não previstos (funcionamento do protocolo ICE). O que era pra ser relativamente simples de ser realizado, acabou sendo o foco principal do projeto.

Foi necessário montar uma grande quantidade de cenários, testes e configurações para tentar identificar o real problema nas ligações entre dois navegadores utilizando a tecnologia WebRTC + SIP. Pesquisas em sites e fóruns especializados, não foi encontrado nenhuma solução definitiva. Desenvolvedores com problemas idênticos, resolviam seus problemas em relação a um

Figura 5.7: Console do Chrome imprimindo informações sobre o uso do protocolo ICE

Fonte: Togo, 2015

cenário específico, não sendo muito útil para este projeto. Simulando um cenário parecido, os resultados não saiu como o esperado. Foi necessário auxílio aos professores para sanar as dúvidas.

Levou um tempo considerável para identificar o problema. Por fim, foi necessário apenas a troca a distrubuição Linux, do Debian para o Ubuntu.

Encontra-se no Apêndice B a configuração completa do Asterisk para o uso do sipML5.

6 *Conclusões*

O projeto possibilitou mostrar na prática o funcionamento do WebRTC com os conteúdos que aprendemos durante o curso, principalmente na matéria de Redes Multimídia, onde se é apresentado os protocolos SIP, RTP e a ferramenta de PABX Asterisk.

A ideia inicial do projeto era apenas montar um cenário dentro da Instituição IFSC-SJ utilizando as tecnologias sipML5 e o Asterisk, e avaliar todo o desempenho desta interação, relatando o funcionamento do sistema, quais plataformas e navegadores o sistema suportava, qualidade do serviço entre outros. Porém, com o decorrer do projeto foram surgindo idéias que poderiam enriquecer a ferramenta. Foram previamente analisadas e depois de alguns estudos e casos de uso, foram adicionadas.

O sistema se mostrou bastante promissor, pois como o WebRTC possibilitou a introdução de um cliente SIP (sipML5) em uma plataforma inteiramente *web*, foi possível desenvolver novas funcionalidades, tornando seu uso rico e interativo aos usuários (uso do chat, identificação dos ramais ativos). Com os desenvolvedores experientes na área e as tecnologias existentes hoje, como o Node.js e o Socket.io, já é possível criar funcionalidades que melhoram a experiência do usuário, tornando-as muito mais robustas. Porém, é necessário um amadurecimento nas documentações que envolvam a interação de um cliente SIP *web* com um PABX IP como o Asterisk ou FreeSwitch. Além de haver dúvidas e complicações durante todo o projeto neste quesito, foi difícil encontrar alguma resposta concreta para os problemas, apenas suposições ou casos bem específicos.

O principal problema do projeto, foi em relação ao funcionamento do protocolo ICE com o sistema, sendo resolvido somente com o auxílio dos professores, onde foram configurados alguns arquivos específicos e a troca da distribuição Linux.

O desenvolvimento das funcionalidades(chat, tempo da ligação, estado do ramal) ocorreram sem problemas, porém houve muito estudo envolvido, pois as tecnologias utilizadas(Node.js, Socket.io), são consideradas recentes.

Num futuro próximo, soluções como estas, ou que tenham algum envolvimento com a

tecnologia WebRTC, tendem a surgir e mudar o modo de como utilizamos vários *softwares* hoje. A tendência será uma migração massificada de várias ferramentas já existentes para uma solução na nuvem.

6.1 Análise dos objetivos do trabalho

- Houve êxito na interação do sipML5 com o Asterisk. Houve problemas de compatibilidade inicialemente, porém ao trocar o sistema operacional, todas as ligações funcionaram normalmente.
- A listagem dos ramais registrados funcionou de maneira eficiente e satisfatória. A aplicação consegue atualizar em tempo real os usuários que foram registrados no Asterisk.
- Os estados dos ramais também é atualizado em tempo real, mostrando quem se conectou ou quem está ocupado em uma ligação.
- O sistema de mensagens instantâneas funcionou normalmente. Apresentou *bugs* em casos bem específicos. Sendo necessário mais tempo para depurá-lo.

6.2 Dificuldades Encontradas

A principal dificuldade enfrentada no projeto foi a parte de interação do Asterisk com o sipML5, gerando razoável atraso na execução do projeto.

Em relação ao desenvolvimento da aplicação, houve bastante estudos e testes durante todo o andamento do projeto, porém grande parte do que foi proposto, foi finalizada com sucesso.

6.3 Trabalhos Futuros

Como sugestão para trabalhos futuros:

1. Testes aprofundados com outras ferramentas que também desepenham o papel de cliente SIP HTML, como por exemplo JsSIP;
2. Implementar outras funcionalidades no sistema, como por exemplo, histório de ligações, alteração manual do estado do ramal, identificação da plataforma que o usuário está conectado, etc.

3. Realizar testes com outra central, como o FreeSwitch. Checar o desempenho no geral, como ligações, captura de informações da central, entre outros.

4. Disponibilizar o uso da ferramenta para redes PSTN, checar viabilidade, desempenho, entre outros.

Referências Bibliográficas

ALMEIDA, P. V. C. de. *Análise de Desempenho de Transmissão de Mídia com Protocolos de Criptografia*. 158 p. Dissertação (Graduação) — Sistemas de Telecomunicações, Instituto Federal de Santa Catarina, Santa Catarina, 2014.

AMARAL, V. M. F. *Framework e Cliente WebRTC*. Dissertação (Mestrado) — Escola de Engenharia, UM, 2013.

BORGES, F. *WebRTC: Estudo e Análise do Projeto*. 56 p. Dissertação (Graduação) — Sistemas de Telecomunicações, Instituto Federal de Santa Catarina, São José, 2013.

GETTING Started with WebRTC. 2014. Disponível em: <http://www.html5rocks.com/en/tutorials/webrtc/basics/>.

HTML5. mar. 2014. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Using_HTML5_audio_and_video>.

INC., G. *WebRTC Website*. jun. 2012. Disponível em: <http://www.webrtc.org>.

JAVASCRIPT. jun. 2014. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.

KHAN, M. *WebRTC Experiments & Demos Website*.

REAL-TIME with Socket.IO and Node.js. jul. 2014. Disponível em: <http://udgwebdev.com/real-time-com-socket-io-no-nodejs/>.

SIP e RTP. jul. 2015. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialpushtotalk2/pagina_3.asp>.

SOUTO WebSockets. jul. 2013. Disponível em: <https://prezi.com/hyv8u1sl6qjx/rtfm-websockets/>.

TRAVERSAL Using Relays around NAT, Relay Extensions to Session Traversal Utilities for NAT. abr. 2010. Disponível em: <https://tools.ietf.org/html/rfc5766>.

VARANDA, J. H. O. *ICE: Uma solução geral para travessia de NAT*. 92 f. Dissertação (Bacharelado) — Faculdade de Ciências da Computação, Universidade de Brasília, Distrito Federal, 2008.

VOZ sobre IP - VOIP. nov. 2011. Disponível em: <http://tele.sj.ifsc.edu.br/ msobral/rmu/slides/aula-22.pdf>.

WEBRTC tutorial using SIPML5. 2015. Disponível em: <https://wiki.asterisk.org/wiki/display/AST/WebRTC+tutorial+using+SIPML5>.

WHAT is a STUN server? 2015. Disponível em: <http://www.3cx.com.br/voip-sip/stun-server/>.

APÊNDICE A – Configurando o lado da aplicação

Neste capítulo, será apresentado a instalação dos principais programas utilizados para montar o ambiente de desenvolvimento da aplicação.

A.1 Apache 2

Se tratando de um sistema *web*, é necessário um servidor que interprete o que foi escrito em código, em formato visual aos usuários. Foi utilizado o Apache 2. Sua instalação é relativamente simples, basta rodar o comando: apt-get install apache2.

```
root@togo-Inspiron-N5010:~# apt-get install apache2
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  asterisk-config asterisk-core-sounds-en asterisk-core-sounds-en-gsm
  asterisk-modules asterisk-moh-opsound-gsm freetds-common gyp
  javascript-common libc-ares-dev libfreeradius-client2 libiksemel3
  libjs-node-uuid liblua5.1-0 libodbc1 libpj2 libpjlib-util2 libpjnath2 libpq5
  libresample1 libsqlite0 libsybdb5 libv8-3.14-dev libv8-3.14.5 node-abbrev
  node-ansi node-archy node-async node-block-stream node-combined-stream
Utilize 'apt-get autoremove' para os remover.
Os pacotes extra a seguir serão instalados:
  apache2-bin apache2-data libaprutil1-dbd-sqlite3 libaprutil1-ldap
Pacotes sugeridos:
  apache2-doc apache2-suexec-pristine apache2-suexec-custom apache2-utils
Os NOVOS pacotes a seguir serão instalados:
  apache2 apache2-bin apache2-data libaprutil1-dbd-sqlite3 libaprutil1-ldap
0 pacotes atualizados, 5 pacotes novos instalados, 0 a serem removidos e 390 não atualizados.
É preciso baixar 1.110 kB de arquivos.
Depois desta operação, 4.708 kB adicionais de espaço em disco serão usados.
Você quer continuar? [S/n]
```

Figura A.1: Instalação do Apache

A.2 Git

Como a aplicação foi baseada no sipML5, foi necessário baixar o projeto original para editá-lo e criar novas funcionalidades. Este estava disponível no GitHub, que é um repositório

de projetos na nuvem, com interação com o programa Git, que é utilizado para o controle de versão de projetos. Ou seja, é possível ter acesso a diversos versões do projeto, a partir do momento em que ela foi salva.

Sua instalação também é simples. É necessário rodar o comando: sudo apt-get install git

```
root@togo-Inspiron-N5010:~# sudo apt-get install git
[sudo] password for togo:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  linux-headers-3.5.0-17
Use 'apt-get autoremove' to remove it.
The following extra packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run git-daemon-sysvinit git-doc git-el git-arch git-cvs git-svn
  git-email git-gui gitk gitweb
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 395 not upgraded.
Need to get 6,824 kB of archives.
After this operation, 15.3 MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Figura A.2: Instalação do Git

A.3 Baixando o código base do sipML5

Após instalado o Git, é necessário clonar do GitHub o projeto sipML5 para a máquina local. É executado o comando git clone, como mostra a Figura abaixo.

```
root@togo-Inspiron-N5010:/opt/lampp/htdocs/togo/testes# git clone https://github.com/sipml5/sipml5.git
Cloning into 'sipml5'...
remote: Counting objects: 2604, done.
remote: Total 2604 (delta 0), reused 0 (delta 0), pack-reused 2604
Receiving objects: 100% (2604/2604), 13.62 MiB | 785.00 KiB/s, done.
Resolving deltas: 100% (1625/1625), done.
Checking connectivity... done.
root@togo-Inspiron-N5010:/opt/lampp/htdocs/togo/testes#
```

Figura A.3: Clonando o projeto do SipML5

A.4 Instalando Node.js

O Node.js neste projeto foi utilizado para duas circunstâncias. Uma foi para subir a aplicação *web* que está utilizando o Sails.js e a outra foi para usar o NPM (Gerenciador de Pacotes do Node) para baixar módulos importantes para o projeto, como o Sails.js e o Socket.io.

```
root@togo-Inspiron-N5010:~# apt-get install nodejs
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  asterisk-config asterisk-core-sounds-en asterisk-core-sounds-en-gsm
  asterisk-modules asterisk-moh-opsound-gsm freetds-common gyp
  javascript-common libc-ares-dev libfreeradius-client2 libiksemel3
  libjs-node-uuid liblua5.1-0 libodbc1 libpj2 libpjlib-util2 libpjnath2 libpq5
  libresample1 libsqlite0 libsybdb5 libv8-3.14-dev libv8-3.14.5 node-abbrev
  node-ansi node-archy node-async node-block-stream node-combined-stream
  node-cookie-jar node-delayed-stream node-forever-agent node-form-data
  node-fstream node-fstream-ignore node-github-url-from-git node-glob
Utilize 'apt-get autoremove' para os remover.
Os pacotes a seguir serão atualizados:
  nodejs
1 pacotes atualizados, 0 pacotes novos instalados, 0 a serem removidos e 389 não atualizados.
É preciso baixar 4.410 kB de arquivos.
Depois desta operação, 62,5 kB adicionais de espaço em disco serão usados.
Obter:1 http://ppa.launchpad.net/chris-lea/node.js/ubuntu/ trusty/main nodejs amd64 0.10.37-1chl1~
trusty1 [4.410 kB]
Baixados 4.410 kB em 8s (515 kB/s)
(Lendo banco de dados ... 247572 ficheiros e directórios actualmente instalados.)
Preparing to unpack .../nodejs_0.10.37-1chl1~trusty1_amd64.deb ...
Unpacking nodejs (0.10.37-1chl1~trusty1) over (0.10.33-1chl1~trusty1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Configurando nodejs (0.10.37-1chl1~trusty1) ...
root@togo-Inspiron-N5010:~# node -v
v0.10.37
```

Figura A.4: Instalação do Node

A.5 Socket.io e Sails.js com NPM

O NPM atualmente vem integrado com o Node, e possibilitou a utilização de duas dependências. O Socket.io, responsável por resolver a parte de lista de ramais e *chat* na aplicação *web*, e o Sails.js, um *framework* MVC que entrega toda uma estrutura de projeto completa e organizada.

Baixando e criando um projeto no Sails.js:

1.sudo npm -g install sails

2.sails new novoProjeto

3.Levantando o servidor, após entrar na pasta novoProjeto: sails lift

Baixando Socket.io:

1.sudo npm install socket.io

A.6 Baixando e executando a aplicação desenvolvida

A ultima versão do projeto desenvolvido, se encontra disponível no GitHub. Basta executar o seguinte comando: git clone https://github.com/togobr/AsteriskAMI_Sailsjs.git.

```
root@togo-Inspiron-N5010:/opt/lampp/htdocs/togo/testes# git clone https://github.com/sipml5/sipml5.git
Cloning into 'sipml5'...
remote: Counting objects: 2604, done.
remote: Total 2604 (delta 0), reused 0 (delta 0), pack-reused 2604
Receiving objects: 100% (2604/2604), 13.62 MiB | 785.00 KiB/s, done.
Resolving deltas: 100% (1625/1625), done.
```

Figura A.5: Clonando projeto desenvolvido

Após clonar a aplicação, é necessário acessar a pasta do projeto e executar o comando: sails lift, para subir a aplicação. Para acessar a página da aplicação, digite: localhost:1337/call, para visualizar o projeto.

```
root@togo-Inspiron-N5010:/opt/lampp/htdocs/togo/AsteriskAMI_Sailsjs# sails lift

info: Starting app...

info:
info:                .-..-.
info:
info:    Sails              <|    .-..-.
info:    v0.11.0              |\
info:                        /|.\
info:                       / || \
info:                     ,'  |'  \
info:                  .-'.-==|/_--'
info:                  `--'-------'
info:    __---___--___---___--___---___--___
info:  ____---___--___---___--___---___--___-__
info:
info: Server lifted in `/opt/lampp/htdocs/togo/AsteriskAMI_Sailsjs`
info: To see your app, visit http://localhost:1337
info: To shut down Sails, press <CTRL> + C at any time.

debug: -------------------------------------------------------
debug: :: Thu Jul 02 2015 23:25:00 GMT-0300 (BRT)

debug: Environment : development
debug: Port        : 1337
debug: -------------------------------------------------------
managerevent { event: 'FullyBooted',
  privilege: 'system,all',
  status: 'Fully Booted' }
```

Figura A.6: Subindo o servidor

APÊNDICE B – Configurando o lado do servidor (Asterisk)

Para que o Asterisk possa ser utilizado com o sipML5, é necessário que seja configurado alguns arquivos para que não haja algum tipo de incompatibilidade e/ou erros na hora de uma chamada. Neste capítulo será mostrado quais as configurações exatas para que o cenário funcionasse com sucesso.

Configuração da máquina utilizada:

- Sistema Operacional Linux - Ubuntu 14.04 LTS
- Asterisk 11.18.0
- IP público - 200.135.233.54
- IP privado - 192.168.1.114

B.1 sip.conf

No arquivo sip.conf, foram feitas algumas alterações. Segue abaixo o que foi adicionado juntamente com a sua configuração inicial.

Importante lembrar que assim como o WebRTC, o uso do SRTP nas últimas versões Google Chrome é obrigatório, fazendo com que as linhas 5, 6, 11, e 12 fossem adicionadas.

O sip.conf também faz referência para outro arquivo chamado contas-sip.conf, através do comando "#include contas-sip.conf"

```
1  [general]
2  udpbindaddr=0.0.0.0:5060
3  realm=200.135.233.54  //Neste campo, adicione o IP publico do seu Asterisk
4  transport=udp,wss,ws
5  tlscertfile=/etc/ssl/certs/asterisk.pem  //Caminho onde foi gerado o seu certificado.
6  tlsprivatekey= /etc/ssl/certs/asterisk.pem
7  tlsdontverifyserver=yes
8  tlsclientmethod=tlsv1
9  localnet=192.168.1.0/255.255.255.0
10 externaddr=200.135.233.54:5060
11 icesupport=yes
12 encryption=yes
```

Figura B.1: Configurações adicionais no arquivo sip.conf

B.2 http.conf

É necessario habilitar a porta para que os módulos res_http_websocket e chan_sip irão utilizar em uma conversa utilizando WebSockets. Observe na Figura B.2.

```
8  [general]
9  enabled=yes
10 bindaddr=0.0.0.0
11 bindport=8088
```

Figura B.2: Configurações no arquivo http.conf

B.3 res_stun_monitor.conf

Para solucionar problemas com NAT, é utilizado um servidor STUN para idenficar o IP público dos usuários conectados. Para isso, é atribuido um endereço do servidor no atributo "stunaddr". Ficará da seguinte maneira:

- stunaddr = stun.l.google.com:19302

B.4 rtp.conf

Configurando as portas para o uso RTP, habilitando o suporte para o protocolo ICE e adicionando novamente o endereço do servidor STUN que será utilizado. Configuração na Figura B.3.

```
6   [general]
7   rtpstart=10000
8   rtpend=20000
9   icesupport=yes
10  stunaddr=stun.l.google.com:19302
```

Figura B.3: Configurações adicionais no arquivo rtp.conf

B.5 contas-sip.conf

O arquivo contas-sip.conf declara e configura todos os ramais dentro do Asterisk. Abaixo um exemplo de declaração de um ramal utilizando o serviço do sipML5.

```
94  [104] ; SIPML5
95  type=friend
96  callerid="Ramal 104" <104>
97  username=104
98  context=ifsc
99  host=dynamic
100 secret=104
101 dtmfmode=rfc2833
102 allowsubscribe=yes
103 subscribecontext=Ramal
104 notifyringing = yes
105 notifyhold = yes
106 limitonpeers=yes
107 rtcachefriends=yes
108 call-limit=10
109 savpf=yes
110 avpf=yes
111 savp=yes
112 transport=udp,ws,wss
113 insecure=port,invite
114 dtlsenable=yes
115 dtlsverify=no
116 dtlscertfile=/etc/ssl/certs/asterisk.pem
117 dtlscafile=/etc/ssl/certs/asterisk.pem
118 dtlssetup=actpass
119 icesupport=yes
120 tlsdontverifyserver=yes
121 encryption=yes
122 directmedia=no
123 videosupport=no
```

Figura B.4: Adicionando um ramal e suas configurações no Asterisk

Configuração de ramais comuns utilizando SIP, há diversos exemplo disponíveis na Internet ou na comunidade do Asterisk.

APÊNDICE C – Lista de ações e eventos do Asterisk

A cada atualização do Asterisk, aumenta o número de informações que podemos coletar, possibilitando uma maior flexibilidade do sistema. Na Figura C.1 apresenta a documentação do Asterisk e alguns exemplos de ações possíveis de se realizar no Asterisk 13. Em vermelho, é possível checar a lista de ações e eventos.

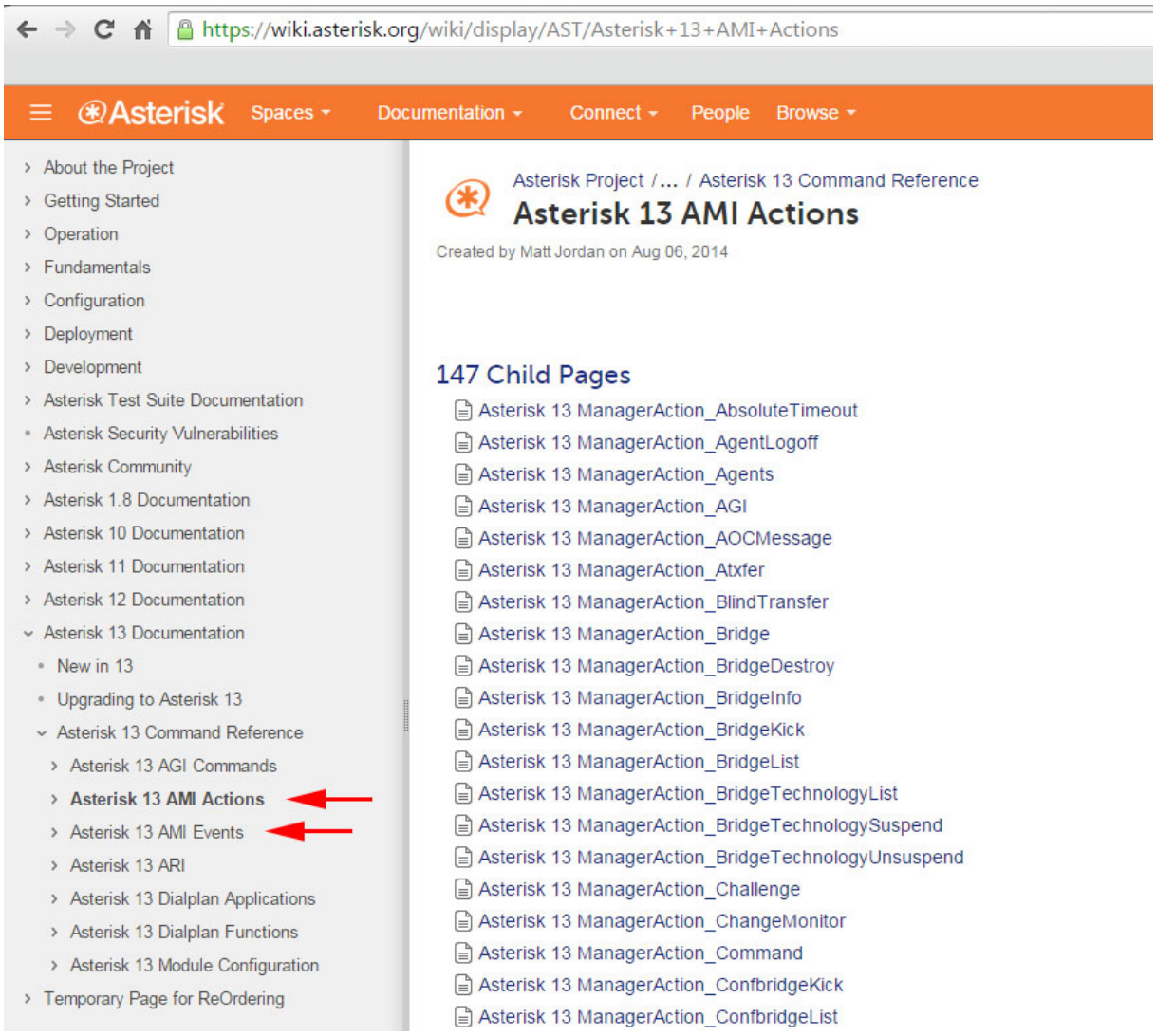

Figura C.1: Documentação do Asterisk. Lista de ações e eventos