GEOCOMMUNICATOR

PATTHAYA PHROMCHEEMUN

PUCHONG POOMBOONTRIK

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEERING

INTERNATIONAL COLLEGE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2013

Thesis – Academic Year 2013

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title : GeoCommunicator

Authors:

1. Miss. Patthaya Phromcheemun Student ID: 53090019
2. Mr. Puchong Poomboontrik Student ID: 53090022

Approved for submission

.....

(Asst.Prof.Dr.Visit Hirankitti)

Advisor

INTERNATIONAL COLLEGE
KMITL

Date August 28, 2014

GeoCommunicator

Miss. Patthaya Phromcheemun 53090019

Mr. Puchong Poomboontrik 53090022

Asst.Prof.Dr.Visit Hirankitti Advisor

Academic Year 2013

ABSTRACT

GeoCommunicator is a web-based communication application, which consists of various types of communication combined with location-based services. The WebRTC technology provides the users with richer communication, which is more powerful than conventional ones. WebRTC works on a browser-to-browser or Peer-to-Peer connection. The purpose of WebRTC is to help create a real-time communication on the web that works across multiple web browsers, both on PCs and mobile devices.

A WebRTC application developed in the thesis supports both audio and video calls including functions for making calls, receiving calls, hanging up calls, chatting, adding/updating/deleting contacts, calling with location services or so-called "GeoCall", doing video conference/meeting, sharing files, and sharing screens. A user can search for a contact in a directory of an organization. The system can detect who is currently talking during the conference meeting and can focus the video on that person. The user can draw notes/diagrams on a shared meeting whiteboard, and he/she can send email to invite other people to join a videoconference.

In this thesis, we also design and develop a contact directory to work with our GeoCommunicator in which we apply a concept of tags and a tree organization structure as the way of contact classification. The tools we used to develop the WebRTC application are HTML5, JavaScript, and WebSocket, for the rest of the development we use Django, a Python web framework, and PostgreSQL to develop it.

Acknowledgements

We would like to express my utmost gratitude and appreciation to our supervisor, Assistant Professor Dr. Visit Hirankitti, who has the attitude and the substance for immense support. His continuous guidance and encouragement despite his busy schedule. Without his guidance and persistent help this project would not have been possible.

Our gratitude is also extended to Dr. Natthapong Jungteerapanich, who helped us to coordinate our project especially in writing this report, and gave the permission to use all required equipment to complete the task for his patience in guiding us throughout the course of our Software Project1.

Our gratitude also goes to all the members in Year's 4 International college students for their warm assistance and lovely friendship.

Table of Contents

Chapter 1

Introduction

In everyday life, most people employ communications technology for their business, transportation and education. The communication on the Internet and wireless network technology supports exchanging of ideas in a convenient way. In this thesis we realize an importance of WebRTC technology, which can provide communication on the web, therefore we develop a web application namely GeoCommunicator using this technology.

A WebRTC application uses VOIP to connect people easily and this saves costs over traditional phone communication. WebRTC enables a web Peer-to-Peer communication using voice and video in real time without requiring any extra plugins.

The browsers that can support WebRTC are Chrome and Mozilla Firefox. The purpose of WebRTC is to help build a real time communication platform that work across multiple web browsers running on PCs and mobile devices.

A WebRTC application is developed for everyone. The WebRTC technology can enhance the ability of communication inside the organization, and enables the organization to reduce costs of voice/video communications substantially.

1.1 Objective

1. To develop a web application by using WebRTC APIs HTML5, JavaScript and Python with the following features:
    1. Videoconference.
    2. Geo-call.
    3. Map location.
    4. Collaborative Drawing.
    5. Contact directory.
    6. Instant messaging.
    7. Collaborative framework of 1-6.
2. To develop GeoCommunicator to run on a PC and a mobile device running a web browser supporting WebRTC.

3. To understand how to configure and set up signaling server.
4. To design a contact directory which works with different sorts of communications relevant to our GeoCommunicator.

1.2 Motivation

We have found that existing voice-over-IP (VOIP) technology is not suitable to run on a web platform; as a result we investigate WebRTC in this thesis in order to solve this problem. WebRTC technology provides a real-time voice and video communication capabilities to run across web platform. It can enhance the ability of communication and save cost because it is flexible to integrate with another web technology.

Web developers can access to WebRTC technology simply by using its JavaScript APIs. A WebRTC application is easier to implement compared with a VOIP desktop application and also there are many Open Source Software that can be used to develop a web application.

As WebRTC is the main technology for the software development in this thesis, it is a good opportunity for us to study and learn its great potential, while we can also learn how to develop a web application using databases, which is new to us.

1.3 Scope of work

The summary of our scope of work is:

- Developing communication tools using WebRTC, that is
    - Videoconference: Demonstrate the communication function. Each client cans video call to each another. Clients can videoconference to each other up to 5 people.
    - Geocall: Client can call to another client directly by click real time icon on the map.
    - Whiteboard: Additional draw collaboration called "Whiteboard". Draw collaboration user can share creative idea by drawing anything to the whiteboard.
    - Instant messaging: Provide instant message to another clients among web browser in real-time.
    - File sharing: Provide file sharing between clients. When client chat they can also sent file directly to each other.

- Screen sharing: Provide screen sharing between clients. When client are in session of conference they can share their own screen window to other client.
- Set up, Install, and Implement Signaling Server and Webserver on real server in Thai Red Cross Society to support WebRTC.

- Developing a contact directory with the following functions and work:
  - Search contact by organization or by employee: User can search by employee or by organization from the directories of Thai Red Cross Society.
  - Implement database on PostgreSQL to store contact of each client for people, company, and government sector.
  - Implement database that can store Tag search in Python Programming Language.
  - Friend feature: user can invite, chat, conference directly to their friends.
  - Collect log data of every activity: The system can collect log data of every activities e.g. chat history and call history.

1.4 Project activities

A diagram of sequential activities as shown below can illustrate the activities we conduct during the project.

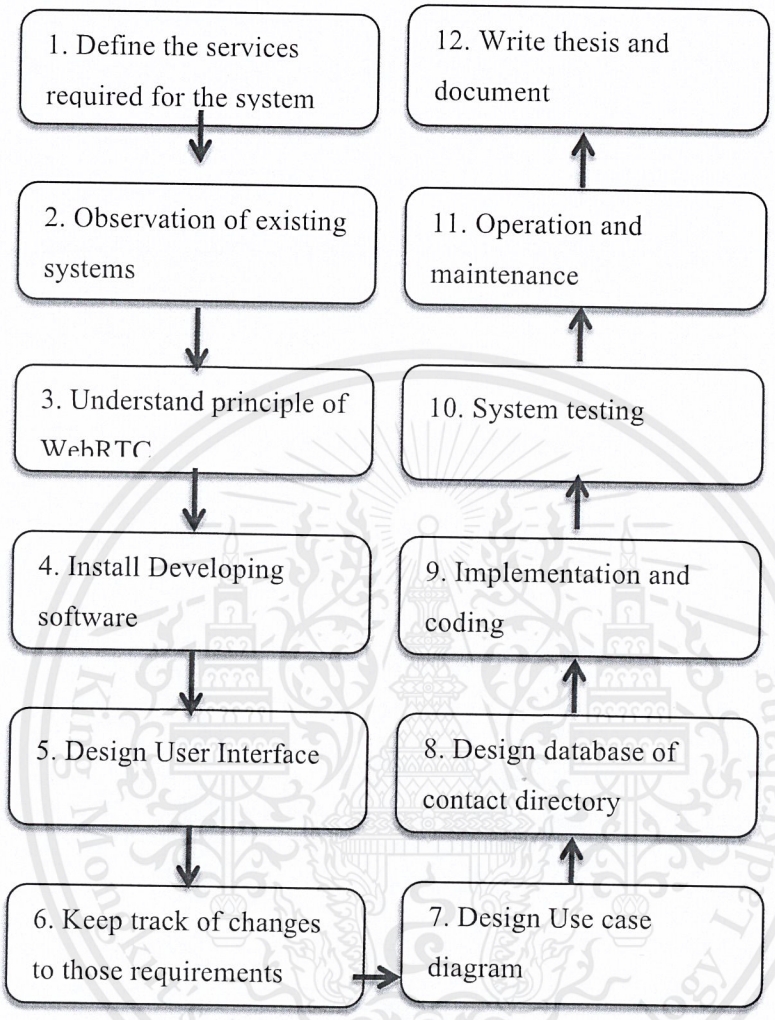


Figure 1.1 sequences of activities of the project

In the beginning, we define what services are required for the system and develop an understanding of the problem domain. Secondly, we observe the existing system, for example, we study video chat features and search contact directories from Skype application then we study the principle of real-time communication and WebRTC Technology. We also study WebRTC APIs and study an example of WebRTC client. Next we install software-developing tools. After this we design a user interface of GeoCommunicator and then keep track of changes to these requirement. Furthermore, we describe the external behavior of the system by using use case diagram. Next we design ER diagram of contact directory in Thai Red Cross Organization. Afterward, we implement the system. Later we do system testing and maintenance. Finally we write a thesis for this software project.

1.5 Structure of this thesis

The sections of the report include:

- Chapter 1 explain the Introduction of the project, Motivation, Objective, Scope of work, Procedure, and Structure of this thesis
- Chapter 2 will explain Problem description and Review of related work
- Chapter 3 explain Background Knowledge the theory and other knowledge relevant to the project
- Chapter 4 Requirements and Analysis explain overall requirements of the system, Relevant system analysis diagrams
- Chapter 5 Software Design explain overall architecture of the software, the functionalities and the design of each component of the system
- Chapter 6 Development explain the development process, technique, and tools that we chose
- Chapter 7 Experimentation illustrates the flow of the usage of our system. Describe the result of the experiment
- Chapter 8 Evaluation and Discussions the effectiveness and the deficiencies of the software developed
- Chapter 9 Conclusion explain summary, lessons learned, problems and obstacles, and further work

Chapter 2

Problem Description and Related Work

2.1 Problem Description

Nowadays VOIP technology is not supported on the web. Developers have to implement it on other platforms. The communications on a web technology can support the exchanging of voice, video and data better than the other communication technologies. WebRTC technology is a communication via the Internet that aims to solve these problems.

In this chapter we will focus on a problem regarding communication tools and review of the related work.

2.2 My proposal

In this project we will investigate the WebRTC technology in order to develop the so-called "GeoCommunicator" which had these properties:

- GeoCommunicator is a communication tool that consist of various features of communication by using map-location information, WebRTC technology, and HTML5 to provide richer user experience and more powerful than normal communication.
- All of these tools can work for collaborative communication.
- Develop communication tool that can cooperate with Geolocation service.
- A web-based communicator for real-time communication of voice, video, short messages, etc. Using WebRTC APIs, JavaScript, Python, HML5
- With its rich contents, open standard, and social-networking capability the web will create a future powerful human communication.
- Develop GeoCommunicator for a PC and a mobile device running a HTML5 web browser-supporting WebRTC.
- Analyze, design contact directory that can work with GeoCommunicator
- Enhance the GIS (Geographical Information System) web service for emergency rescue with WebRTC real-time communication in Thai Red Cross Society Organization.

2.3 Related Work

Currently, VOIP applications enable users to communicate on the Internet with features such as file sharing, group conference calling, messaging, as well as one-to-one communication. Although this technology is developed for PCs and mobile devices, it is not supported on the web technology.

2.3.1 Skype

Skype is primarily a VOIP service and the most popular free voice and audio communication service in the world. Being different from other VOIP services, Skype is a hybrid peer to peer and client-server system. Skype is changing how people communicate with the integration of voice and instant messaging over the Internet into one application. Talk when you want and type when you want, and switch between the two as you wish.

The service allows users to communicate with peers by voice using a microphone, video by using a webcam, and instant message. Calls to other users within Skype are free of charge, you pay nothing more than the monthly Internet service, and while calls to landline telephones and mobile phones are charged. Skype allows user to know, if a friend is offline, when she is willing to communicate. Skype also include other features such as file transfer and videoconference.

The Skype software can be downloaded freely from www.skype.com. Users need to install Skype application on their computers, or smartphones to begin making VOIP calls. Skype clients are supported on Windows, OS X, iOS, Android, BlackBerry, Kindle, smart TV's, X-Box, and PlayStation. User can update a new version of Skype to get the latest features and improvements.

Figure 2.1 Skype's features

We observed the contact directory feature from Skype application. The ability to search Skype's contact is a powerful feature.

- User can search by a Skype name, full name, or e-mail address, user can narrow their search based on a set of filters.
- Use the search for Skype User box to enter a full name, enter an e-mail address, and enter a Skype name. Search by location.

- If user don't know a name or e-mail address, user can try searching by location. Select a country or region from drop-down list. Enter a city-state, or both by entering the names into the boxes provided.
- Search by language; select their language from drop-down list. Language searches work best in combination with other parameters unless users are looking for something very specific.
- Search by gender or age range. Search people in Skype Me mode; Skype can display online status, and Skype Me mode is an online status that lets other people know you are ready to receive call and chat from anyone. Skype Me mode box guarantees that people that user find are people who want to talk.

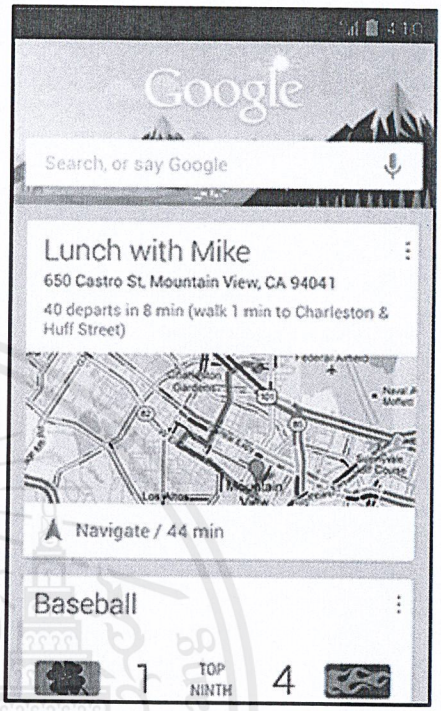
2.3.2 Line

Line is an free application for instant messaging service with VOIP technology on iOS, Android, Windows Phone, PC and BlackBerry that can be used between iOS and Android as well as used between other operating systems. Main properties of Line include instant message, free voice or video calls with high quality and at any time you can talk as much as you want, send video/voice message easily, send stickers and emoticons to their friends, customize wallpaper, group chat (create and join groups where users can chat), play games, add friends/contacts. Later Line is added with home and timeline features making communication more funny and convenient. Examples of Home and timeline features are update of text status, pictures, videos, and sending current or any specific locations with the highlight on stickers, which display emotions.

In this thesis we studied the style and how to add contacts from Line which describe as follows:

- Adding contacts from the list on the user's mobile phone: If there are friends who are using Line application, show Line symbol then one can add them as friends immediately.
- QR code: scanning the QR code to add friends/contacts. User can create own QR code for other people to scan QR code for adding friend in Line.
- Shake it: User shakes the phone in case of two user's phone are together when they shake at the same time, they can be friends.
- Search by ID: can find friend from Line ID by typing Line ID of friend.

2.3.3 Google Now



Google Now is a service and an application in Android operating system since the release of Android 4.1 Jelly Bean; it was developed by Google as a private secretary to learn user behavior and present daily advice to users without user is command. It tries to predict what the information the user wants while the user does not need to tell it and it tries to offer a short explanation of the spot in the form of a new interactive interface called the card, which is similar to a widget. As Google is the leader of web search service so the results are always acceptable. For instance, for image search, it will show keyword immediately without having to search on the web. It learns from the user how to begin to work, and travel through a place on the road. By the time user go out to work, Google Now will alert the user that it is the time to work now and tell the traffic along the road while the user is driving to work.

The format of Google Now cards:

- Traffic conditions base on the current and passed location. The user can specify the routes where user needs to know such as a route to go to work
- Public transportation such as when the next bus will come. When users are getting near to a bus stop, Google Now tells them what the bus schedules are.
- Next appointment and next event using data from Google Calendar
- Flights; departure time, gate or building to go
- Sport results in real time and fixture of the next match. User can also buy or reserve tickets as well
- Location information; when users are outside, Google Now will recommend and show restaurant, bars and places where the user can go
- Weather reports of current location and other parts of the world
- Translation from Google translate
- Currency converter
- Local time and current time of other cities

Chapter 3

Background Knowledge

In this chapter we will explain the background knowledge that used for development of GeoCommunicator, which includes WebRTC, Signaling Server, Django Web framework, and map location services.

3.1 Introduction to WebRTC

WebRTC is a real-time multimedia communication technology fully integrated on web browser in peer-to-peer via simple JavaScript APIs without installing any plugins.

For the first time, browsers will interact directly with other browsers. A real-time audio and video call between two browsers is possible. Communication in such a scenario might involve direct media streams between the two browsers.

The sequence of interactions involves following:

- The caller browser and the caller JavaScript application, through the JavaScript APIs.
- The caller JavaScript application and the application provider (Web Server).
- The application provider and the receiver JavaScript application
- The receive JavaScript application and the receiver browser (again through the application-browser JavaScript API).

WebRTC API Process:

1. Get User Media: Get User Media is available in Chrome, Opera and Firefox that allows a web browser to access the camera and microphone. It represented a single source of synchronized either audio or video. We use the getUserMedia() method in JavaScript to access to these local media devices.
2. Peer Connection: Peer Connection is the top level API for WebRTC that set up between two or more browsers and set up the audio/video calls. It is used to establish a connection to the other clients and, it can be able to send data across the network. After we create getUserMedia() method, we do this step to make a connection between 2 clients. When the other side received the data, it will be coming up with a new media stream for the getUserMedia() method.

3. RTC Data Channel: RTC Data Channel is similarly to the WebSocket of JavaScript APIs. It's used to send non-media data across the network to exchange data. We can choose unreliable or reliable for the data.

3.1.1 How WebRTC works within a web browser

Figure 3.1 shows the browser model and the real-time communication function. The RTC function interacts with the web application using standard APIs. It communicates with the Operating System using the browser. WebRTC is the interaction that occurs browser-to-browser called Peer Connection where the function in one browser communicates using on-the-wire standard protocols with the RTC function in another browsers. While web traffic uses TCP for transport, the on-the-wire protocol between browsers can use other transport protocols such as User Datagram Protocol (UDP). The signaling Server provides the signaling channel between the browser and the other end of the Peer Connection.

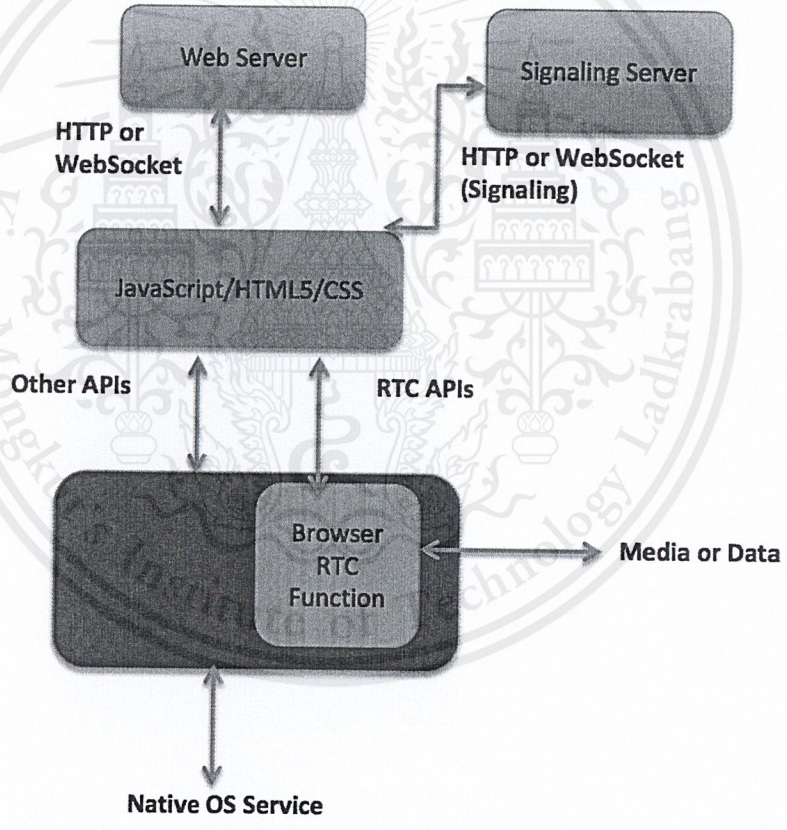


Figure 3.1 WebRTC in the Browser

The set of elements in WebRTC includes web servers, browsers running various operating systems on many devices including desktop PCs, tablets, and mobile phones, and other server. WebRTC enables communication among all these devices. A Peer Connection establishes the transport for voice and video media and data flows directly between the browsers. While we sometimes refer to the connection between the browser and signaling

server, it is not really signaling as used in telephony systems. Signaling is not standardized in WebRTC as it is just considered part of the application. This signaling may run over HTTP or WebSocket to the same web server that serves HTML pages to the browser, or to a completely different web server that just handles the signaling.

3.1.2 Communication among multiple devices

WebRTC supports multi-party or conferencing sessions involving multiple browsers. The way to do this; have each browser establish a Peer Connection with the other browsers in the session. This is shown in Figure 3.2. Each browser establishes a full mesh of Peer Connection with the other browsers. For audio, this might mean mixing the media received from each browser. For video, this might mean mixing the media received streams from other browsers to different windows with appropriate labeling. As new browsers join the session, new Peer Connection is established to send and receive the new media streams.

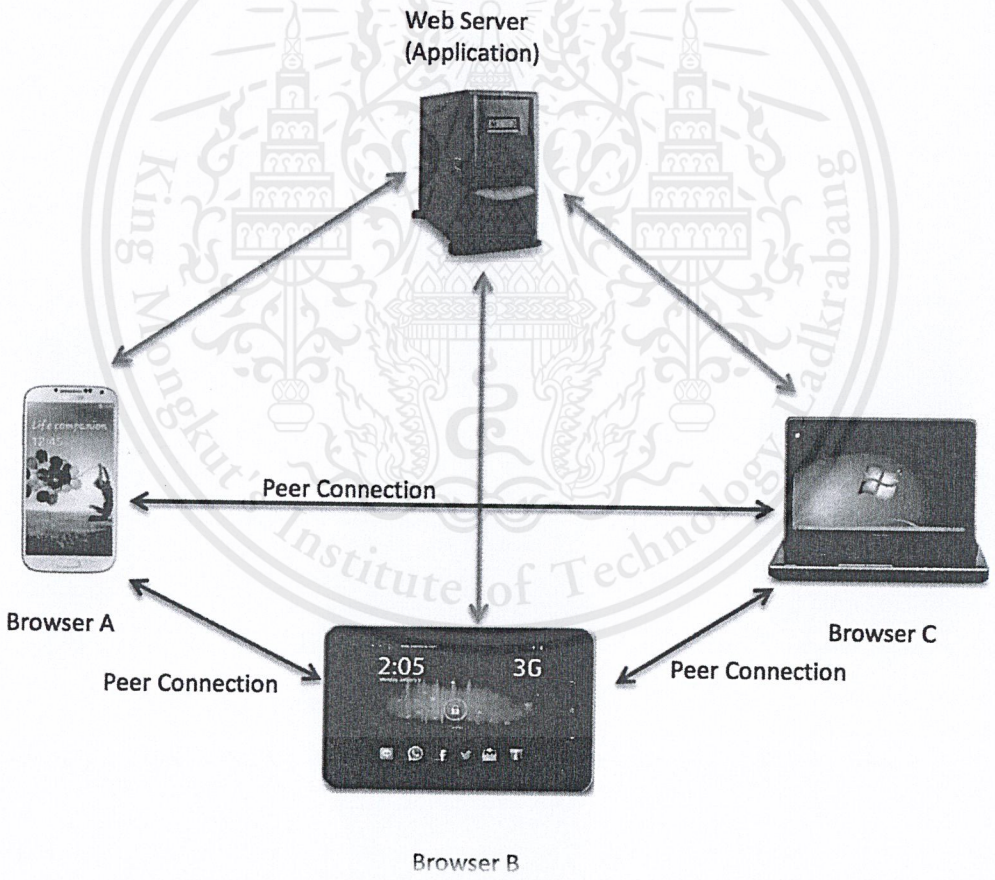


Figure 3.2 Multiple Peer Connections

WebRTC enables peer-to-peer communication.

WebRTC still need a server for clients to exchange metadata and to coordinate communication, this is called "signaling" this is cope with network address translators (NAT) and firewalls. In this section we will explain how to create a signaling service and the mechanism to do with the real-world communication by using STUN and TURN servers. In addition we explain the method how WebRTC handle multiple calls and interacts with services such as VOIP.

Signaling is the process of coordinating communication for a WebRTC application to set up a 'call' that clients use to exchange the following information:

- Session control messages.
- Error messages.
- Codecs and codec settings, bandwidth and media types.
- Key data to establish secure connections.
- Host's IP address and port.

The reason why signaling is not defined by WebRTC is to avoid redundancy and maximize compatibility using established technologies, therefore signaling methods and protocols are not specified by WebRTC standards.

3.1.3 Signaling security

Encryption is needed for all WebRTC components. However, signaling mechanisms are not defined by WebRTC standards, If an attacker manages to hijack signaling, they can stop sessions, redirect connections and record, alter or inject content, so it depends on you to make signaling secure.

The most important thing in securing signaling is to use secure protocols, such as HTTPS and WSS (i.e. TLS), which ensure messages cannot be intercepted unencrypted. Also be careful not to broadcast signaling messages in a way that other callers using the same signaling server can access them. To secure a WebRTC signaling can uses TLS (transport layer security).

3.1.4 Building a signaling service with Socket.io on Node

- Using Node.js
  - Node.js Its uses concept of a server side JavaScript asynchronous event-driven model, which is good to work with real time.
- Using socket.io
  - Socket.IO is a JavaScript library for real-time web applications.
  - It has two parts: a client-side library that runs in the browser, and a server-side library for node.js it is event-driven on server side.
- Using "Signaling Server" Software
  - A simple signaling server for clients to connect and do signaling for WebRTC.
  - Specifically created as a default connection point for SimpleWebRTC.js from SimpleWebRTC.com
- How to Setup and run Signaling Server

  (1) Install node.js (2) Install socket.io (3) Running "Node Server.js" on terminal

3.2 Django Python Web Framework

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It was designed to make common websites, web applications and web service. We learn about MVC web framework, which is written in Python, a powerful and popular programming language.

The MVC pattern of the web development consists of

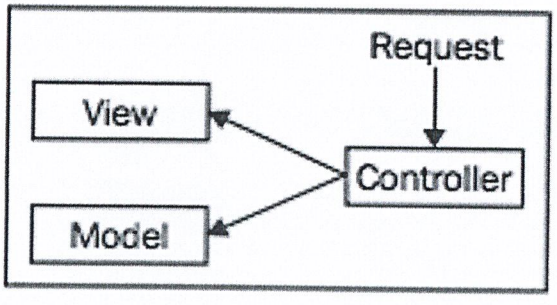


Figure 3.3 component of the MVC pattern

- Models: Describes your data structure/database schema.
- Views: Controls what a user sees.
- Templates: How a user sees it.
- Controller: The Django Framework, URL parsing.

3.2.1 why we adopt Django as our Web framework

The reason why we use Django as the web framework to change GeoCommunicator is the following:

- The web framework is based on Python, which is our language chosen for server side system development.
- Django lets us divide codes into logical groups or MVC pattern to make it flexible to change
- It provides auto generated web admin to ease the website administration
- It provides pre-packaged API for common user tasks
- It provides you templates system to define HTML templates for your web pages to avoid code duplication

3.3 Map Location Service

Location Services are web services which make use of location information. Depending on our device and available services, Location Services uses a combination of Internet network and Geographic Information System to determine user location on the screen.

WebRTC application can show user location including Maps, indicate user current location using a marker. Maps, directions, and location-based depend on data services. These data services are subject to change and may not be available in all geographic areas, resulting in maps that may be unavailable, inaccurate, or incomplete.

The Geographic Information System is a powerful tool, which provides an excellent means to effectively support planning, and decision-making processes. GIS has been being used widely, especially on website application to provide location services. In this project, we apply GIS technology with website application to provide emergency service. The application includes two main components, which are friendly map viewer and GIS tools. Tools are

provided for user to interact with the map in order to request for rescue such as zooming, panning the map, pin location on the map to send requests.

We use HTML5 to provide user interface and JavaScript language to develop the application. There are two libraries of JavaScript that play the main role in this project which are OpenLayers for creating map and providing tools, and also JQuery for handling event, animation and manipulation.

Our software focuses on how to apply GIS and WebRTC technology to help providing a better emergency service of the Thai Red Cross organization in order to help saving people's lives.

OpenLayers is an open source JavaScript library for viewing map data on a web browser. It provides an API for building rich web-based geographic applications similar to Google Maps and Bing Maps. The library was originally developed based on the Prototype JavaScript Framework.

Geo-location is the identification of the real-world geographic location of an object, such as radar, mobile phone or an Internet-connected computer terminal. Geo-location allow a web application to access current location of the device running the web browser.

Chapter 4

Requirement Analysis

WebRTC application supports audio and video calls including the function of chat, add/update/delete contact, Geo-call, create conference room, join conference room, share files, share screen. Client can search contact by organization or employee. The system can detect who are talking in conference mode and focus on video of that person. Draw/Erase Whiteboard. Send Email; client can send email to invite other people to join videoconference.

4.1 Requirements

4.1.2 System Functional Requirements

1. Implement WebRTC application to supports audio and video calls, Conference.

2. The system can communicate two-way or conference real-time communication between web browser and web browser, instant messaging among web browser.

3. The system will automatically locate the current position of user.

4. User can find near friend on map.

5. The system can add, update and delete contact.

6. System allow user to find contacts.

7. User can videoconference to each other up to 5 people.

8. The user can use Geo-call to another users by click real time icon on the map to invite friend to communicate.

9. User can search person in organization by Tag search, search for organization to communicate with. For example Fire station, Police station.

10. User can share screen and share file between the users

11. The system can detect who are talking in conference mode and focus on video of that person.

12. The system can use on cellular or 3G, 4G network.

13. User can draw or paint picture together to share ideas easily.

- Draw area call whiteboard

- Whiteboard can draw/erase line, shape, change size of line, change Colors, typing text in whiteboard
- User can download picture to theirs device

14. User can send short message to invite in conference to their friends.

15. User can call directly by using phone number in directories

4.1.3 Non-System Functional Requirements

1. Using HTML5 to implement GUI that contains functions:

- The application allows user to create and join Room for Conference.
- The application shows screen room that already created.
- Application shows the screen of input box and allow user to typing message and show screen of conversation
- Application shows the screen of video calling that contain self view and remote view while user is calling
- Implement interface login and logout form

2. Using Web socket and HTML5 canvas to implement whiteboard function.

3. Session and data of videoconference, chat message, collaboration drawing must be secured.

4. Video conference must work across platform.

4.2 System Analysis

4.2.1 Use case Diagram

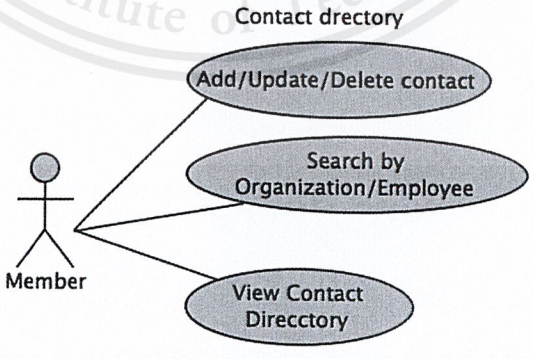


Figure 4.1 Use case Diagram Contact Directory

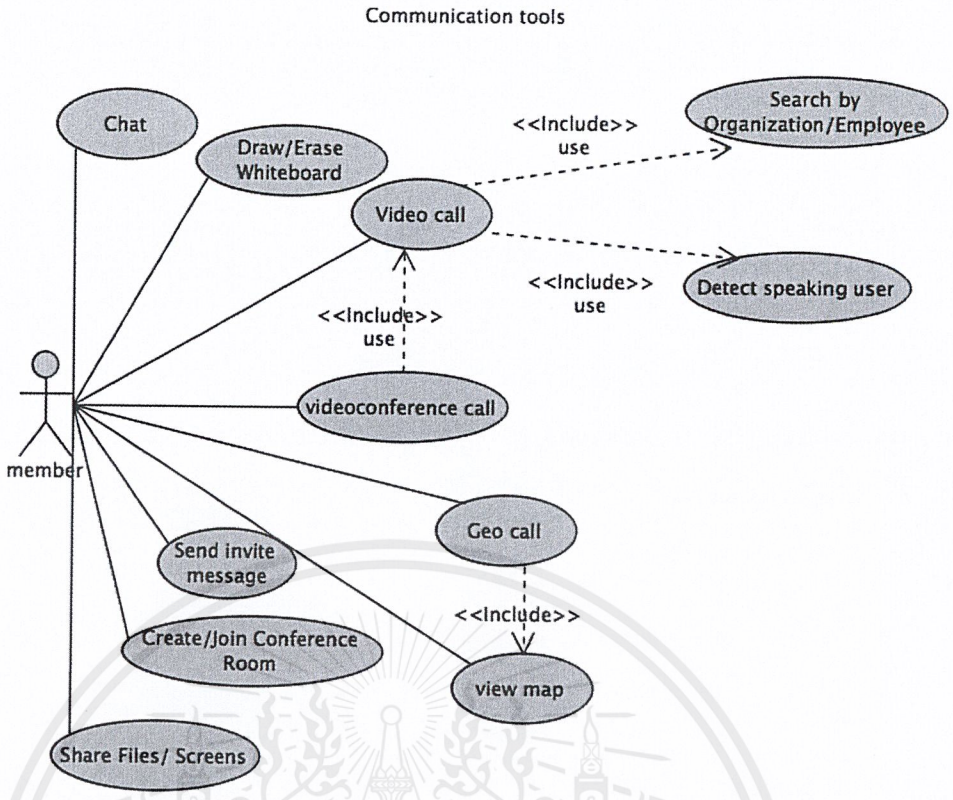


Figure 4.2 Use case Diagram Communication tools

4.2.2 Use case Description

GeoCommunicator consists of 2 parts: Contact Directory and Communication tools

Use case: Log in Actor: Client Overview: This use case describes the first time that client need to use WebRTC application, the client must log in into the system by input username and password. Next click login button. If the clients do not have username and password, they can register before login into the system.

Use case: Log out Actor: Client Overview: This use case describes when the client has finished, they can log out by clicking log out button.

Use case: Add/Update/Delete contact

Actor: Client

Overview: This use case describes a client this use case describes Client add the information of contact into the system. Their details including first name, family name, phone number, address, email, and job position. The client can add, update, view, search, and delete contact. . If client wants to delete a contact, he simply chooses the remove option.

Use case: Make call

Actor: Client

Overview: This use case describes a client call to another client. WebRTC supports audio and video calls, Conference.

Use case: Receive call

Actor: Client

Overview: This use case describes allow client to answer calls at another clients.

Use case: Hang up call

Actor: Client

Overview: This use case describes if the conversation is ended or the client gets unwanted call, the client can press Hang up button.

Use case: Chat

Actor: Client

Overview: This use case describes if the clients who are already log in, they can chat among web browser to each other.

Use case: Draw/ Erase whiteboard

Actor: Client

Overview: Client can draw or paint picture together to share ideas easily. Whiteboard can draw/erase line, shape, change size of line, change colors, typing text in whiteboard

Use case: Geo Make call

Actor: Client

Overview: This use case describes a client call to another client directly by click real time icon on the map. WebRTC supports audio and video calls and Conference.

Use case: Geo Receive call

Actor: Client

Overview: This use case describes allow client to answer calls at another clients.

Use case: Geo Hang up call

Actor: Client

Overview: This use case describes if the conversation is ended or the client gets unwanted call, the client can press Hang up button.

Use case: Send Email

Actor: Client

Overview: This use case describes client can send email to other client.

Use case: Detect speaking user

Actor: Client

Overview: This use case describes the system can detect who are talking in conference mode and focus on video of that person.

Use case: Create Conference Room

Actor: Client

Overview: This use case describes in case of calling more than two people client can conference to communication. Client can create room to wait other people to join the session of room conference.

Use case: Join Conference Room

Actor: Client

Overview: This use case describes when the client get session key from owner room, they can join directly to the conference room.

Use case: Share Files

Actor: Client

Overview: This use case describes when client chat, they can also sent file directly to each other.

Use case: Share screen

Actor: Client

Overview: This use case describes when clients are in session of conference; they can share their own screen window to other clients.

Use case: Search by employee/organization

Actor: Client

Overview: This use case describes client can search by employee or by organization from the directories of Thai Red Cross Society.

Chapter 5

Software Design

5.1 System architecture of GeoCommunicator

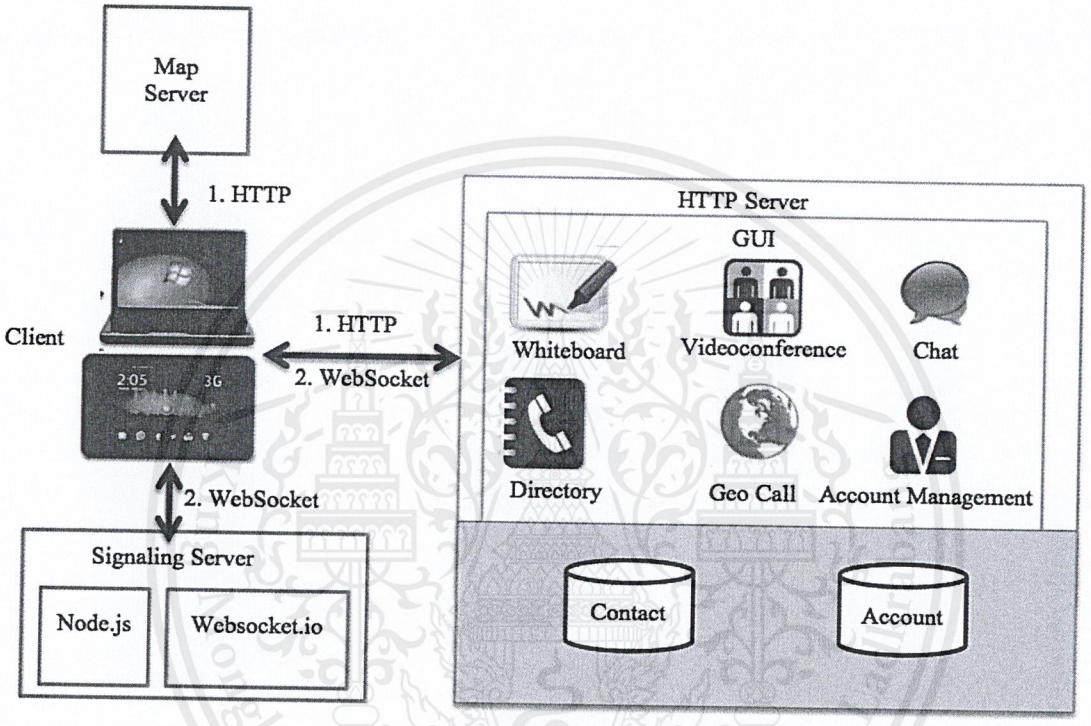


Figure 5.1 System architecture

GeoCommunicator contains 2 main parts, which are communication tools and databases. The detail of each part is described below.

- Communication tools
  - Videoconference: Demonstrate the communication function. Each client cans video call to each another. Clients can videoconference to each other up to 5 people.
  - Chat: Provide instant message to another clients among web browser in real-time.

- Whiteboard: Additional draw collaboration called "Whiteboard". Draw collaboration user can share creative idea by drawing anything to the whiteboard.
- Geocall: Client can call to another client directly by click real time icon on the map.

- Databases
  - Contact directory database: Search contact by organization or by employee: User can search by employee or by organization from the directories of Thai Red Cross Society.
  - Account database: Management of account

System architecture description

Client as a user member that login to web application using HTTP/HTTPS protocol then user can view contact list of directory database.

User can use communication tools such as videoconference, chat, draw whiteboard, view contact in map, each time they communicate to other user they will using WebSocket protocol to register to signaling server that can locate user IP address, then user will use WebSocket to transport data to each other.

In case of using map we will sent data of geo location to map-server by HTTP protocol and view where are we, to make decision which other friend in map we want to communicate with.

5.1.1 Communication flow of the system

In practice most devices many live behind more layers of NAT, some of them have anti-virus software that blocks certain ports and protocols, and many of them are behind proxies and corporate firewalls. The similar device, such as a home Wi-Fi router, may in fact implement a firewall and NAT.

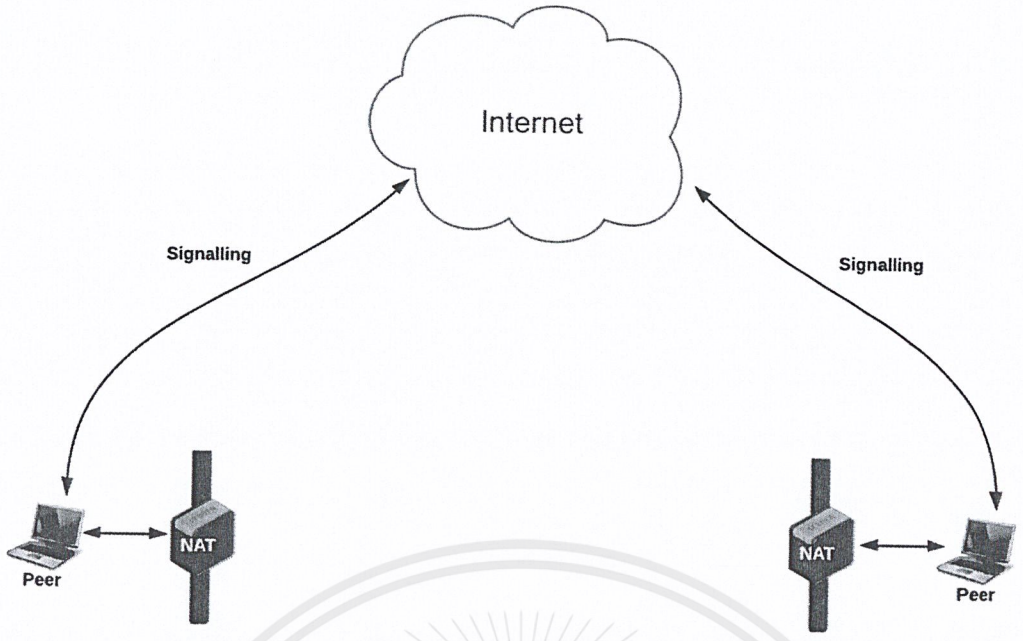


Figure 5.2 Peer-to-Peer Client a communicate with Client B

WebRTC apps can use the **ICE framework** to overcome the complexities of real-world networking. To make this happen, your application must pass ICE server URLs to RTC Peer Connection, as described below.

ICE tries to find the best path to connect peers. It tries all possibilities in parallel and chooses the most efficient option that works. ICE first tries to make a connection using the host address obtained from a device's operating system and network card; if that fails ICE obtains an external address using a STUN server, and if that fails, traffic is routed via a TURN relay server.

STUN server is used to get an external network address. TURN servers are used to relay traffic if direct (peer to peer) connection fails.

5.1.1.1 STUN

STUN servers are on the public internet and have one simple task is to check the IP:Port address of an incoming request and send that address back as a response. This process enables a WebRTC peer to get a publicly accessible address for itself, and then pass that on to another peer via a signaling mechanism, in order to set up a direct link. WebRTC calls successfully make a connection using STUN but though this can be less for calls between peers behind firewalls and complex NAT configurations.

5.1.1.2 TURN

RTCPeerConnection tries to set up direct communication between peers over UDP. If it fails, RTCPeerConnection resorts to TCP. If that fails, TURN servers can be used as a fallback, relaying data between endpoints. TURN is used to relay audio/video/data streaming between peers, not signaling data.

TURN servers have public addresses, so peers can contact them even if the peers are behind firewalls or proxies. They have a conceptually simple task to relay a stream, but unlike STUN servers they inherently consume a lot of bandwidth.

The diagram below shows TURN in action: pure STUN didn't succeed, so each peer resorts to using a TURN server.

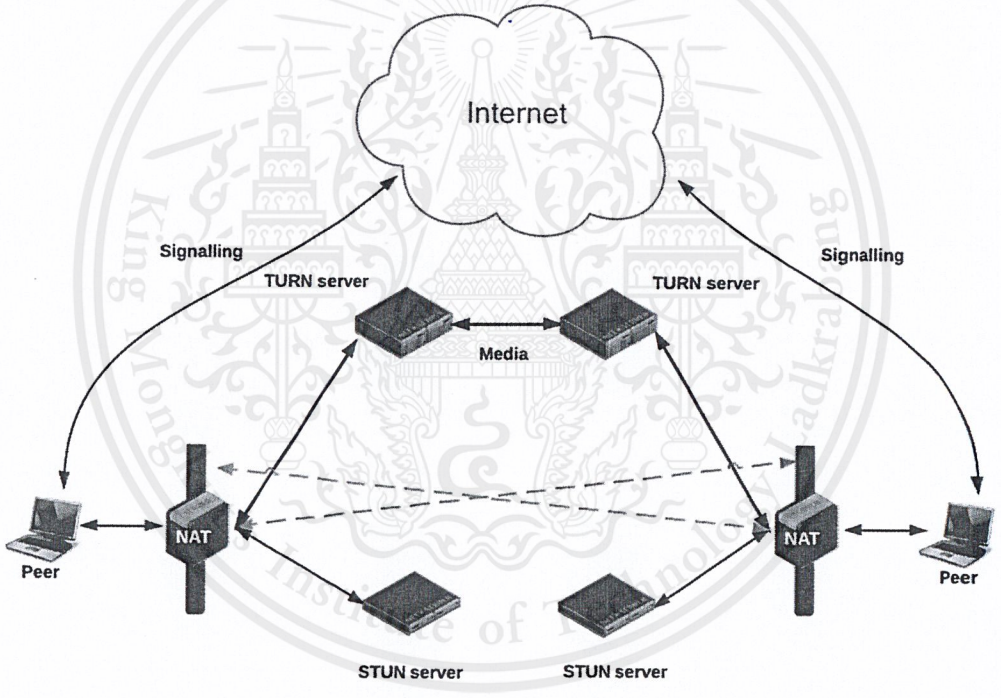


Figure 5.3 the full Monty: STUN, TURN and signaling

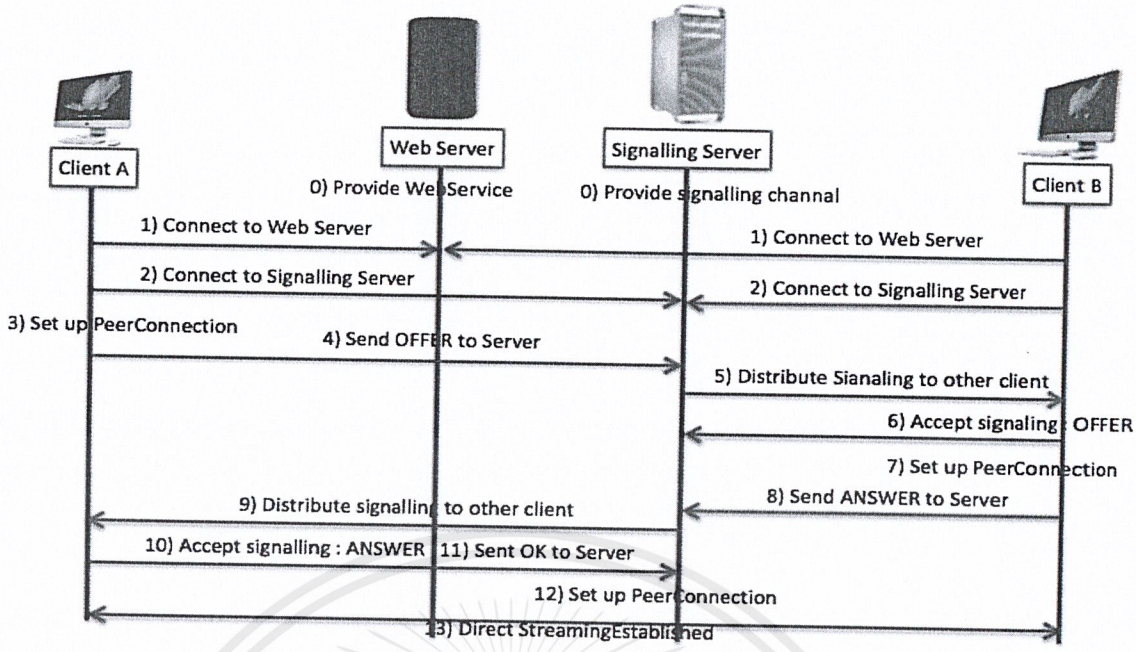


Figure 5.4 Peer-to-Peer Client a calling Client B

Figure 5.4 Peer-to Peer Client a calling Client B Description

1. Step web server and signaling server are running service. Web server contains the web content,
2. Signaling server provide signaling channel for WebRTC client.
3. Suppose Client A want to call to Client B. client A will connect to web server to download HTML and JavaScript then JavaScript file is automatically connected to Signaling server to register their IP address.
4. When Client B connect to web server to download HTML and JavaScript same as Client A. Signaling server will know that every Clients are online then Client A will set up Peer Connection then send OFFER to Signaling server.
5. Signaling server will distribute signaling to other Clients (Client B).
6. Client B will be accepted Signaling: OFFER and set up Peer Connection then send ANSWER to signaling server.
7. Next, when Signaling server gets answer from Client B, the server will distribute Signaling to Client A. Client A accept signaling: ANSWER then send OK to Signaling to server.
8. Finally, Signaling will provide Peer Connection for Client A and B. And the data between Client A and Client B will directly be streaming established

5.2 Design of contact database

The design of contact directory we collect data of organization as hierarchy.

Collect word of each organization as tags to search later. Tags searching aim to help user easier to find tight result.

5.2.1 Hierarchy

Logically a hierarchy can be modeled as a rooted tree: the root of the tree forms the top level, and the children of a given vertex are at the same level, below their common parent. A child node does not allow being a parent of a node on a high level. To implement this, a hierarchy can be modeled using a graph or a pre-order relation on the set of items. Alternatively, items of like type can be grouped together, and the hierarchy can be modeled using a partial order relation on the set of sets-of-like-items.

5.2.2 Tags

A **tag** is a keyword or term assigned to a piece of information (such as an Internet bookmark, digital image, or digital content). This kind of metadata helps describe an item and allows it to be found again by browsing or searching. Tags are generally chosen informally and personally by the item's creator or by its viewer, depending on the system.

We use the "Objected-Oriented" Concept to create data model of contact database in Django. From hierarchy of Thai Red Cross Organization Chart we start from the top level to the end of level.

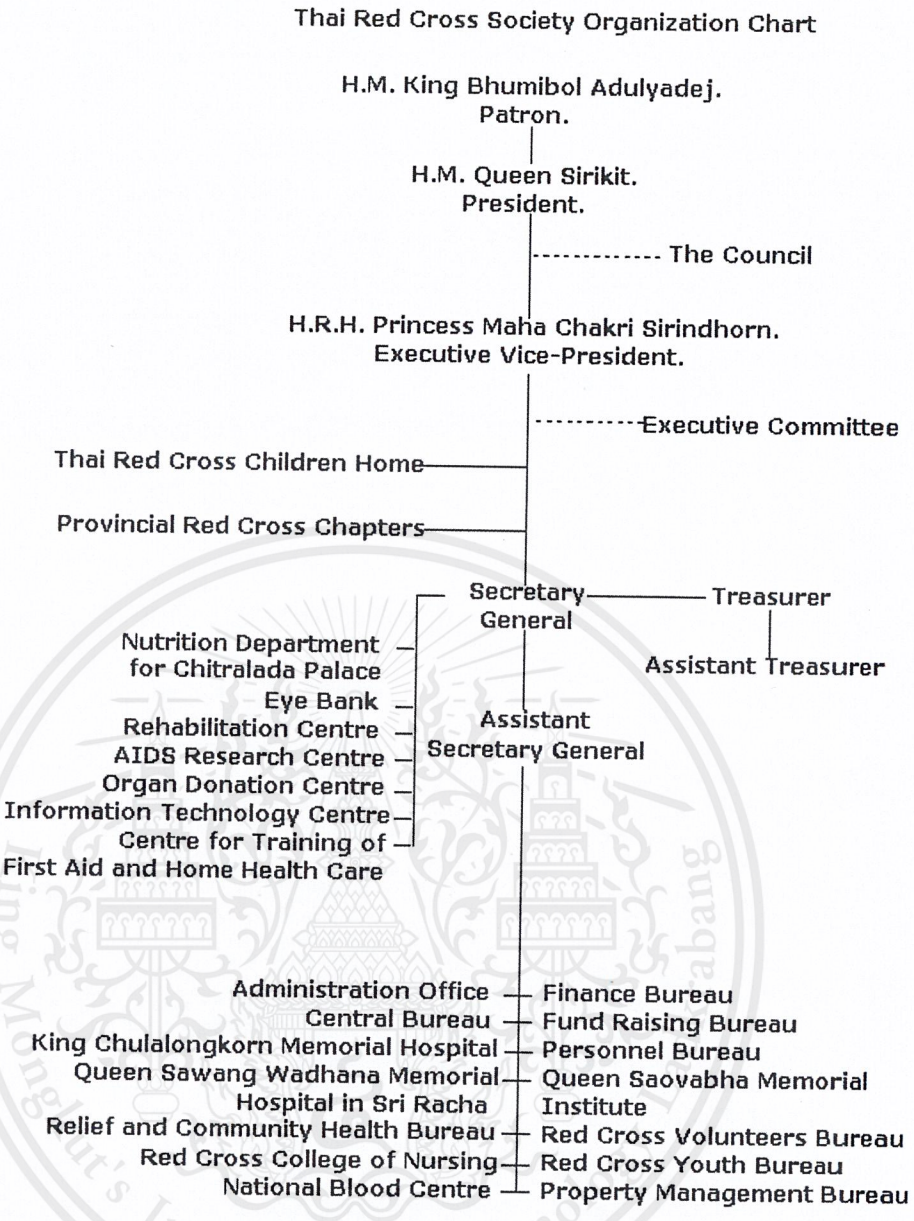


Figure 5.5 Thai Red Cross Organization Chart

5.3 Class Diagram

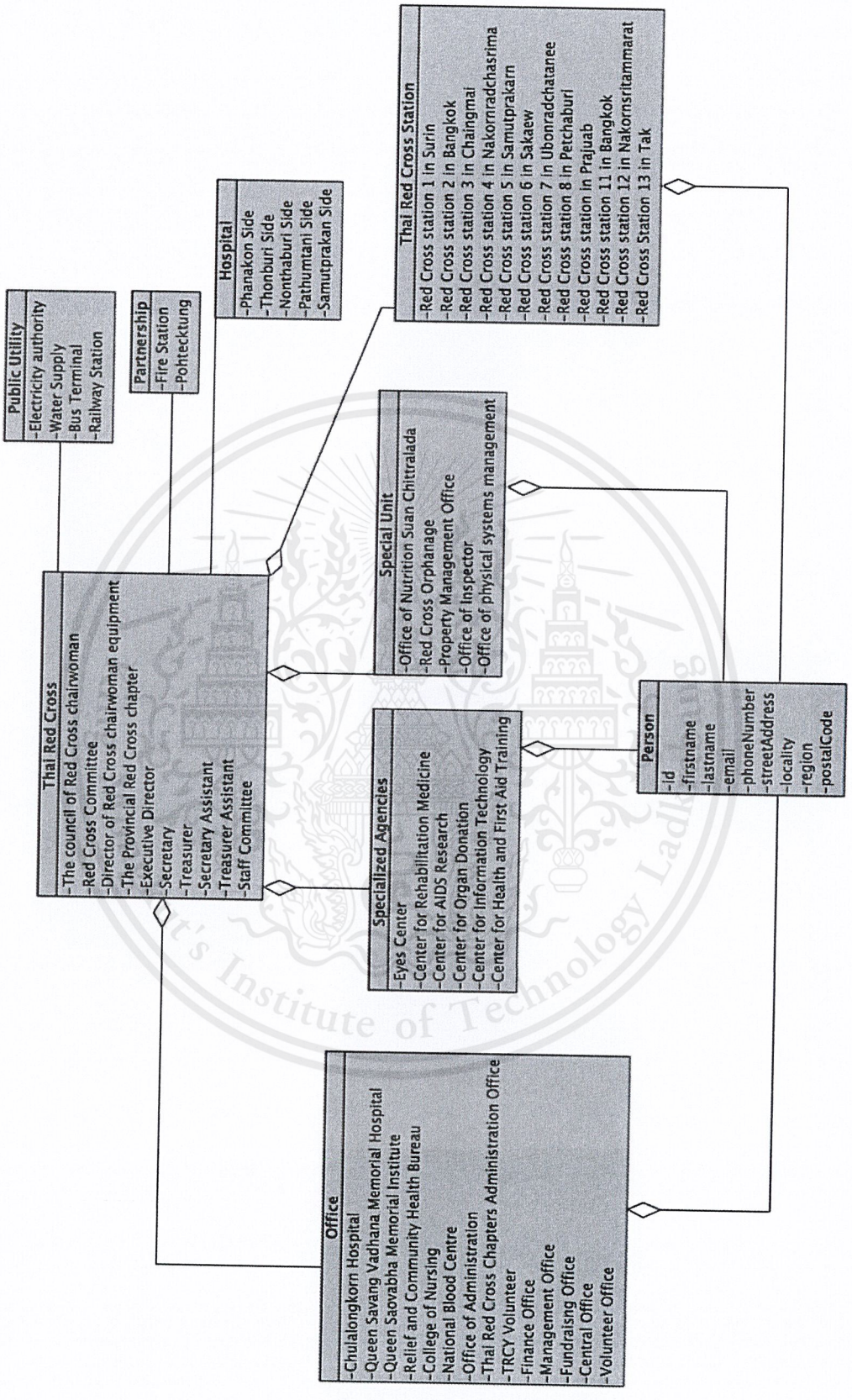


Figure 5.6 Class Diagram of Thai Red Cross Society Organization Chart

5.3.1 Class Diagram Description

Class "Thai Red Cross"

The top level is "Thai Red Cross" contains attributes; The council of Red Cross chairwoman, Red Cross Committee, Director of Red Cross chairwoman equipment, The Provincial Red Cross chapter, Executive Director, Secretary, Treasurer, Secretary Assistant, Treasurer Assistant, Staff Committee. This class contains class "Office", "Specialized Agencies", "Special Unit" and "Thai Red Cross Station". This class associates with class "Public Utility", "Partnership", "Hospital".

Class "Public Utility"

This class contains attributes; Electricity authority, Water Supply, Bus Terminal, Railway Station.

Class "Partnership"

This class contains attributes; Fire Station, Pohtecktung.

Class "Hospital"

This class contains attributes; Phanakon Side, Thonburi Side, Nonthaburi Side, Pathumtani Side, Samutprakan Side.

Class "Office"

This class contains attributes; Chulalongkorn Hospital, Queen Savang Vadhana Memorial Hospital, Queen Saovabha Memorial Institute, Relief and Community Health Bureau, College of Nursing, National Blood Centre, Office of Administration, Thai Red Cross Chapters Administration Office, TRCY Volunteer, Finance Office, Management Office, Fundraisng Office, Central Office, Volunteer Office. This class contains class "Person".

Class "Specialized Agencies"

This class contains attributes; Eyes Center, Center for Rehabilitation Medicine, Center for AIDS Research, Center for Organ Donation, Center for Information Technology, Center for Health and First Aid Training. This class contains class "Person".

Class "Special Unit"

This class contains attributes; Office of Nutrition Suan Chittralada, Red Cross Orphanage, Property Management Office, Office of Inspector, Office of physical systems management. This class contains class "Person".

Class "Thai Red Cross Station"

This class contains attributes; Red Cross station 1 in Surin, Red Cross station 2 in Bangkok, Red Cross station 3 in Chaingmai, Red Cross station 4 in Nakornradchasrima, Red Cross station 5 in Samutprakarn, Red Cross station 6 in Sakaew, Red Cross station 7 in Ubonradchatanee, Red Cross station 8 in Petchaburi, Red Cross station in Prajuab, Red Cross station 11 in Bangkok, Red Cross station 12 in Nakornsritammarat, Red Cross Station 13 in Tak.

Class "Person"

The end of level contains attributes; id, -firstname, lastname, email, phoneNumber, streetAddress, streetAddress, region, postalCode.

Chapter 6

Development

In this chapter we will describe technique to implement each component in system, including tools and resource that involve in development process.

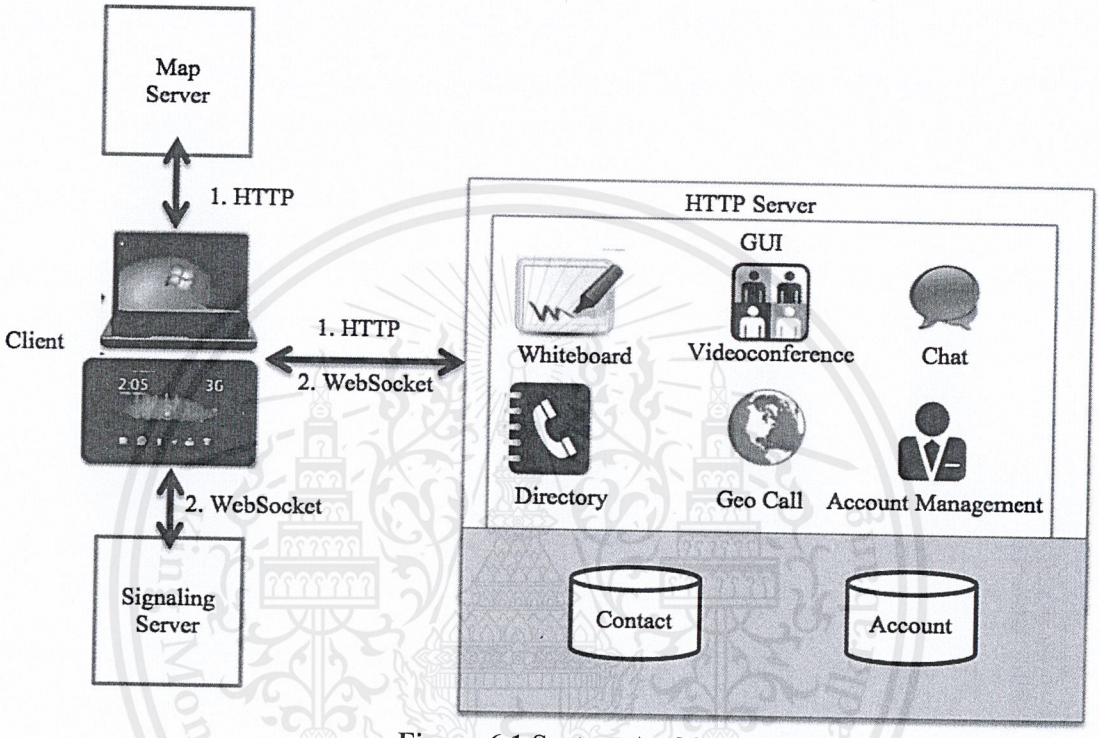


Figure 6.1 System Architecture

6.1 Web application development

GeoCommunicator consists of 2 component; client side application and server side application.

6.1.1 Client Side Application

- HTML5: It is a markup language used for structuring and presenting content on the web browser. HTML5 contains many features to support for the latest multimedia and also WebRTC technology.
- JavaScript is used to develop our client-side application running on a web browser
- SimpleWEBRTC.js is a JavaScript library to implement the WebRTC

- The lists of web browser that supported WebRTC application
    - PC
        - Google Chrome more than version 23
        - Mozilla Firefox more than version 22
    - Android
        - Google Chrome more than version 28
        - Mozilla Firefox more than version 24

6.1.2 Server Side Application

- Signaling Server:
    - Using **Socket.io** is a JavaScript library for real-time web applications. It has two parts: a client-side library that runs in the browser, and a server-side library for node.js it is event-driven.
    - **Node.js**: Contains a built in HTTP server library to run a web server without the use of external server and uses the concept of a server side JavaScript asynchronous event-driven model, which is suitable to work with Real time.
    - Signal Master
- In Ubuntu Server 12.04 LTS OS.
- Database:
    - Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design.
    - PostgreSQL is Open Source Database

6.2 Implementation of communication functions

6.2.1 Videoconference

```
function openroom()
{
window.open(location.href);
}
var room = location.search &&
location.search.split('?')[1];
var valname =$('#nameInput')
$('#sessionInput').val("room")
valname.val("owner")


function room_created()
{
       var btn=document.createElement("BUTTON");
       var t=document.createTextNode(location.search &&
location.search.split('?')[1]);

       btn.appendChild(t);
       document.body.appendChild(btn);
};

// create our webrtc connection
var webrtc = new SimpleWebRTC({
       // the id/element dom element that will hold "our" video
       localVideoEl: 'localVideo',
       // the id/element dom element that will hold remote videos
       remoteVideosEl: 'remotes',
       // immediately ask for camera access
       autoRequestMedia: true,
       debug: true,
       detectSpeakingEvents: true,
       autoAdjustMic: false
});

// when it's ready, join if we got a room from the URL
webrtc.on('readyToCall', function () {
       // you can name it anything
       if (room) webrtc.joinRoom(room);
});

function setRoom(name) {
       $('form').remove();
       $('h1').text('Room :'+ name);
       //$('#subTitle').text('Link to join: ' + location.href);
       $('body').addClass('active');
       $("#slide_container_show > #slide_container > #slide_page > #phone >
#contact-list > li > sip").text('Room : '+name);
       //$("#slide_container_show > #slide_container > #slide_page > #phone >
#contact-list > li > name").text(valname.val());
       //room_created();
}

if (room) {
       setRoom(room);
} else {
       $('form').submit(function () {
              var val = $('#sessionInput').val().toLowerCase().replace(/\s/g,
'-').replace(/[^A-Za-z0-9_\-]/g, '');
              webrtc.createRoom(val, function (err, name) {
                     console.log(' create room cb', arguments);

```

Get room name for create session from URL that follow by "?" operator.

Create our WebRTC connection

Use to join conference room when stay in the same URL

```
                        var newUrl = location.pathname + '?' + name;
                        if (!err) {
                                history.replaceState({foo: 'bar'}, null, newUrl);
                                setRoom(name);
                        } else {
                                console.log(err);
                        }
                });
                return false;
        });
}
// Screen sharing
var button = $('#screenShareButton'),
        setButton = function (bool) {
                button.text(bool ? 'share screen' : 'stop sharing');
        };

setButton(true);

button.click(function () {
        if (webrtc.getLocalScreen()) {
                webrtc.stopScreenShare();
                setButton(true);
        } else {
                webrtc.shareScreen(function (err) {
                        if (err) {
                                setButton(true);
                        } else {
                                setButton(false);
                        }
                });

        }
});

```

Function of screen sharing

6.2.2 Function for Chat

```
<script>
        // grab the room from the URL
                var room, nick, query,
                $messages = $('#messages');

                if (location.search) query =
location.search.split('?')[1].split('&');
                if (query && query.length) {
                        for (var i=0, l=query.length, item, key, val; i<l; i++)
{
                                item = query[i] && query[i].indexOf('=') !== -1 ?
query[i].split('=') : null;
                                if (item) {
                                        key = item.shift();
                                        if (item.length) val = item.shift();
                                        if (key === 'room') room = val;
                                        if (key === 'nick') nick = val;
                                }
                        }
                }
                function renderChat(message) {

        $messages.append($('<li/>').addClass('message').html('<span>' +
message.from + ':</span> ' + message.msg));
                }
                if (room) {
                        $('#room, #room_label').hide();
```

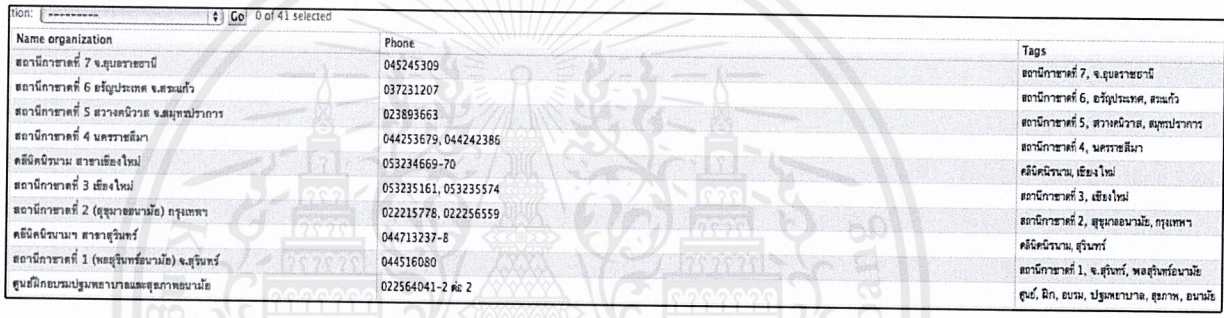
```
                    $('#room_btn').text('Set Nickname');
                    $('h1').text('Chatting
in'+room+'as'+(nick||'[Anonymous]'));
                }
              var webrtc = new SimpleWebRTCData({debug: true});

              webrtc.on('readyToCall', function () {
                if (room) webrtc.joinRoom(room);
              });

              webrtc.on('data', function (message) {
                console.log('message ' + message);
                renderChat(message.payload);
              });

              function joinRoom(name, nick) {
                $('body').addClass('active');
                $('h1').text('Chatting in ' + room + ' as ' + nick);
                $('.join_link').text('Others can join here: ' + location.href);
              }

              if (room && nick) {
                joinRoom(room, nick);
              } else {
                $('.room_form').submit(function (e) {
                      var room_set = !!room;
                      room = $(this).find('#room').val() || room;
                          nick = $(this).find('#nick').val();
                      console.log('room name is ' + room);

                      if (!room_set) {
                          webrtc.createRoom(room, function (err, name) {
                                console.log(' create room cb', arguments);

                            var newUrl = location.pathname + '?room=' + name;
                              if (!err) {
                            history.replaceState({foo: 'bar'}, null, newUrl);
                                joinRoom(room, nick);
                              } else {
                                console.log(err);
                                if (err === 'taken') {
                                    webrtc.joinRoom(room)
                                }
                              }
                          });
                      } else {
                          joinRoom(room, nick);
                          webrtc.joinRoom(room);
                      }

                      return false;
                });
              }

              $('textarea').on('keydown', function (e) {
                if (e.keyCode && e.keyCode === 13) {
                      var $this = $(this),
                            msg = $this.val(),
                            payload = {from: nick, msg: msg};
                      renderChat(payload);
                      $this.val('');
                      webrtc.sendToAll('data', payload);
                      return false;
                }
              });
</script>
```

6.3 Django data model implementation

6.3.1 Organization – models.py

```
1   class Thai_Red_Cross(models.Model):
2       Thai_Red_Cross_id = models.AutoField(primary_key=True)
3       name_organization = models.CharField(max_length=100)
4       phone = models.CharField(max_length=100)
5       fax = models.CharField(max_length=100)
6       location = models.CharField(max_length=100)
7       tags = models.CharField(max_length=100,null=True,blank=True)
8
9       def __unicode__(self):
10                  return u'%s' % (self.name_organization)
```

Thai_Red_Cross Model consistes of Thai_Red_Cross_id, name_organization, phone, fax, location, tags filed which use to store the Thai Red Cross Organization information.



tion: ---------- Go 0 of 41 selected

Name organization	Phone	Tags
สถานีกาชาดที่ 7 จ.อุบลราชธานี	045245309	สถานีกาชาดที่ 7, จ.อุบลราชธานี
สถานีกาชาดที่ 6 อรัญประเทศ จ.สระแก้ว	037231207	สถานีกาชาดที่ 6, อรัญประเทศ, สระแก้ว
สถานีกาชาดที่ 5 สวางคนิวาส จ.สมุทรปราการ	023893663	สถานีกาชาดที่ 5, สวางคนิวาส, สมุทรปราการ
สถานีกาชาดที่ 4 นครราชสีมา	044253679, 044242386	สถานีกาชาดที่ 4, นครราชสีมา
คลินิคนิรนาม สาขาเชียงใหม่	053234669-70	คลินิคนิรนาม, เชียงใหม่
สถานีกาชาดที่ 3 เชียงใหม่	053235161, 053235574	สถานีกาชาดที่ 3, เชียงใหม่
สถานีกาชาดที่ 2 (สุขุมาลอนามัย) กรุงเทพฯ	022215778, 022256559	สถานีกาชาดที่ 2, สุขุมาลอนามัย, กรุงเทพฯ
คลินิคนิรนามฯ สาขาสุรินทร์	044713237-8	คลินิคนิรนาม, สุรินทร์
สถานีกาชาดที่ 1 (พลสุรินทร์อนามัย) จ.สุรินทร์	044516080	สถานีกาชาดที่ 1, จ.สุรินทร์, พลสุรินทร์อนามัย
ศูนย์ฝึกอบรมปฐมพยาบาลและสุขภาพอนามัย	022564041-2 ต่อ 2	ศูนย์, ฝึก, อบรม, ปฐมพยาบาล, สุขภาพ, อนามัย

Figure 6.2 Database table created by Django

Admin inputs all of Thai Red Cross Organization information and split the name of Thai Red Cross Organization into tags fields. The example; สถานีกาชาดที่ 7 จ.อุบลราชธานี (name organization) = "สถานีกาชาดที่ 7", "จ.อุบลราชธานี" (Tags).

6.3.2 Tags Model in Tree Hierarchy – models.py

```
1   class Genre(MPTTModel):
2       name = models.CharField(max_length=50 , unique=True)
3       parent = TreeForeignKey('self', null=True, blank=True
4   related_name='children')
5
6       def get_absolute_url(self):
7           return reverse('genre_detail', kwargs={'pk': self.pk, })
8
9       def __unicode__(self):
10          return '%s%s' % ('-' * self.level, self.name)
11
12      class MPTTMeta:
13          order_insertion_by = ['name']
14
```

Genre class model consists of name, and parent fields to store parent and children node such as parent node = " สถานีกาชาด" children node = "นครราชสีมา", "สระแก้ว"

6.3.3 Show Tree hierarchy – genre_list.html

This code uses to implement tags with hierarchy.

```
{% recursetree object_list %}
          <ul>
                <li>
                    <a href="#">{{ node.name }}</a>

                    {% if not node.is_leaf_node %}

                        <a href="#">{{ children }}</a>

                    {% endif %}
                </li>
          </ul>
{% endrecursetree %}
```

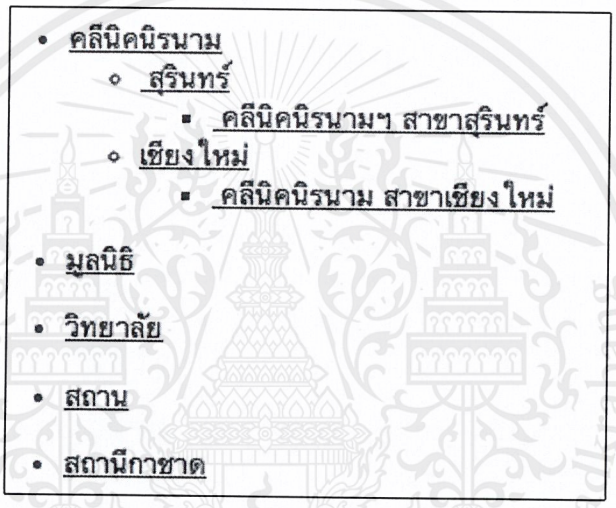


Figure 6.3 Tree hierarchy tag in Thai Red Cross

6.3.4 Employee Model – model.py

Employee model consists of person_id, firstname, lastname, job_position, mobile, organization, and friend field which use to store employee's information in Thai Red Cross Organization.

```
class Employee(models.Model):
    person_id = models.AutoField(primary_key=True)
    firstname = models.CharField(max_length=100)
    lastname = models.CharField(max_length=100)
    job_position = models.CharField(max_length=100)
    mobile = models.CharField(max_length=100)
    organization = models.CharField(max_length=100,null=True,blank=True)
    friend = models.CharField(max_length=100,null=True,blank=True)

    def __unicode__(self):
                return u'%s %s' % (self.firstname, self.lastname)

class EmployeeForm(ModelForm):
    class Meta:
        model = Employee
```

นางแสงอุรา	หมูเยื้อ	นักการภารโรง	0896648811
นายสุชาติ	เกาแดง	เจ้าหน้าที่ธุรการ	0865229359
นางสาวศรีสุดา	เชาวลิต	เจ้าหน้าที่การเงินและบัญชี	0804486936
นางอรนุช	วิมลศิริ	เจ้าหน้าที่การเงินและบัญชี	0870528197
นางกิตติยา	จิตต์เจียม	เจ้าหน้าที่การเงินและบัญชี	0817787264
นางสาวอภินยา	นุชนาถ	เจ้าหน้าที่การเงินและบัญชี	0865775409
นางสาวกัญญาณัฐ	บัวแย้ม	เจ้าหน้าที่การเงินและบัญชี	0861457234
นางสาวณัฐปภัสร์	ครุฑโต	เจ้าหน้าที่การเงินและบัญชี	0879178566
นางสาวสุรีพร	โหนชุนทด	เจ้าหน้าที่การเงินและบัญชี	0896289961

Figure 6.4 The Thai Red Cross employees information

6.4 Implementation of supporting function

6.4.1 Tags Function in view.py

```
class MyCustomUpdateView(UpdateView):
    model = Genre

    def get_form_kwargs(self):
        """
        Returns the keyword arguments for instanciating the form.
        """
        kwargs = super(MyCustomUpdateView, self).get_form_kwargs()
        kwargs.update({'my_first_param_to_init_form': 1,
                       'my_second_param_to_init_form': 2,
        })
        return kwargs

```

6.4.2 Organization tags search – view.py

Tags search organization function. Input output

```
def search(request): #organization
    error = False
    if 'q' in request.GET:
        q = request.GET['q']
        if not q:
            error = True
        else:
            thai_Red_Cross =
Thai_Red_Cross.objects.filter(name_organization__contains=q)
            return
render_to_response('Directories/index.html',{'Thai_Red_Cross':
thai_Red_Cross, 'query': q})

            #return
render_to_response('Directories/search_result.html',{'Thai_Red_Cross':
thai_Red_Cross, 'query': q})
            #return render_to_response('Directories/index.html',{'error':
error})
    return render_to_response('Directories/index.html',{'error': error})
```

6.4.3 Organization tags search result – search_result.html

```
<p>You searched for: <strong>{{ query }}</strong></p>
{% if Thai_Red_Cross %}
     <ul>
        {% for i in Thai_Red_Cross %}
        <li>
         Name:{{ i.name_organization }}
         <br>Phone:{{ i.phone }}
         <br>Fax: {{ i.fax }}
         <br>Location: {{ i.location }}
             </li>


        {% endfor %}
    </ul>
{% else %}
    <p>No matched your search criteria.</p>
{% endif %}
```

6.4.4 General user create contact directories function – view.py

Showing a new blank form for adding employee information. The information consists of first name, last name, job position, and mobile phone. Next, saving form data provided by the user to the associated model. If there are errors, redisplay a new blank form for adding employee information again.

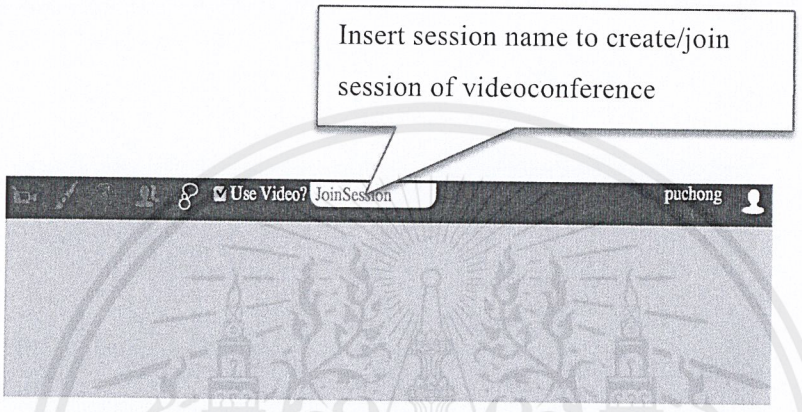
```
def create_employee(request):
if request.method == 'POST':
        post = request.POST

        firstname = post['firstname']
        print firstname

        lastname = post['lastname']
        print lastname

        job_position = post['job_position']
        print job_position

        mobile = post['mobile']
        print mobile

        f = EmployeeForm(request.POST)
        if f.is_valid():
                new_employee = f.save()
                return
                HttpResponseRedirect(reverse('Directories.views.index_thai')
                )
return
render_to_response('Directories/create_employee.html',context_instance=Requ
estContext(request))
```

Chapter 7

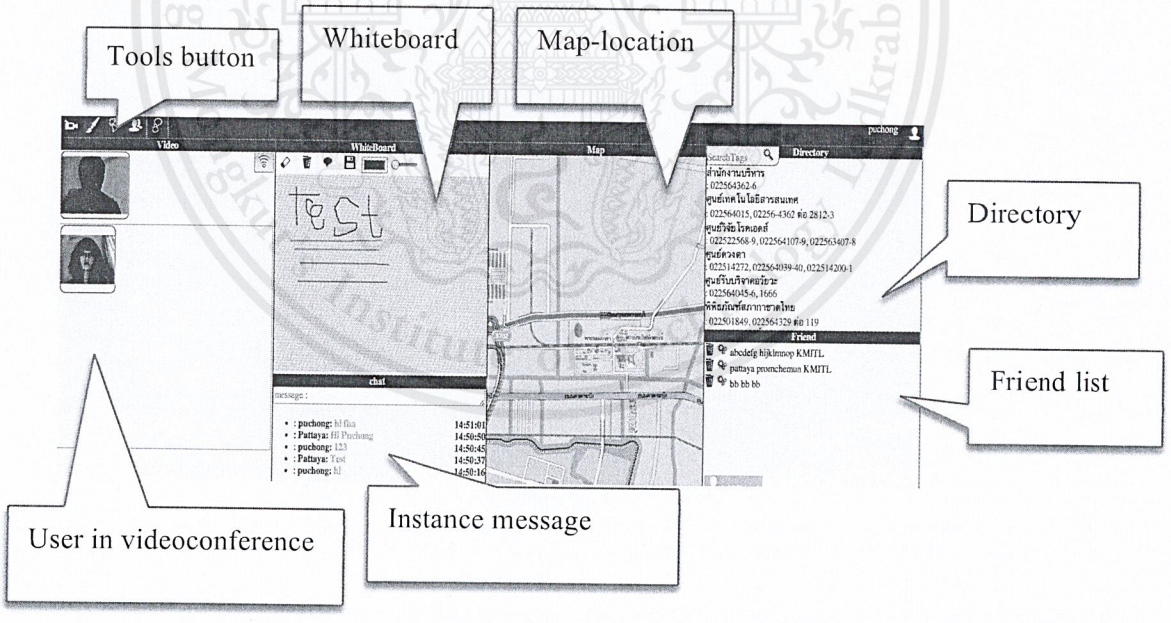
Experimentations

7.1 Functionalities of GeoCommunicator

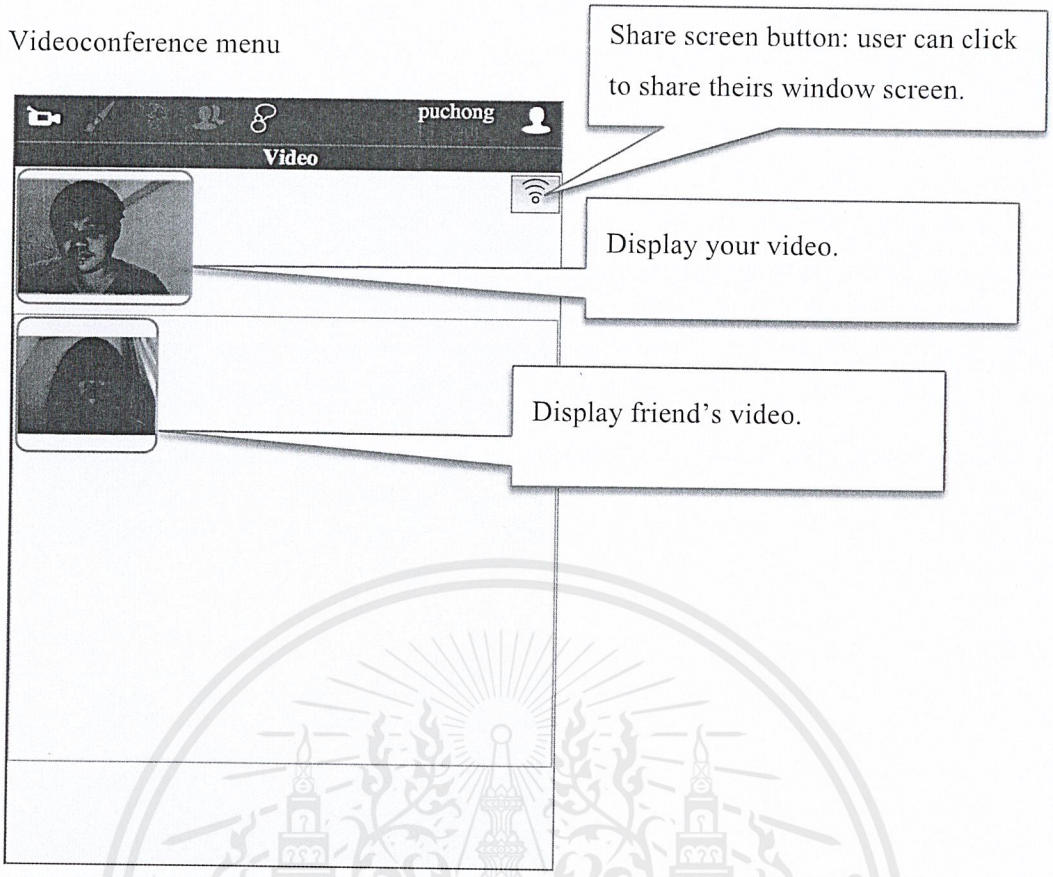
This is the main page of GeoCommunicator web application, which contains user came to website with no communication session all tools will be hidden as picture below.



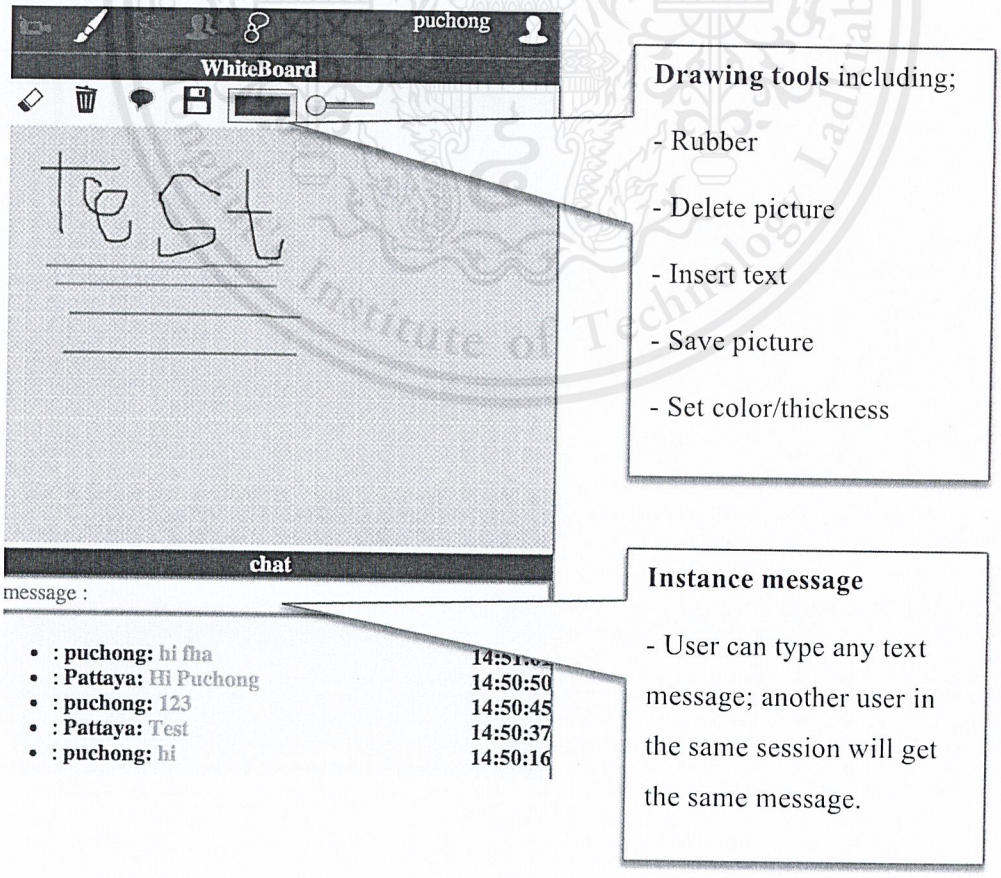
- Main page with communication session all tools will visible. User can show and hide each tool by click on tools button icon.



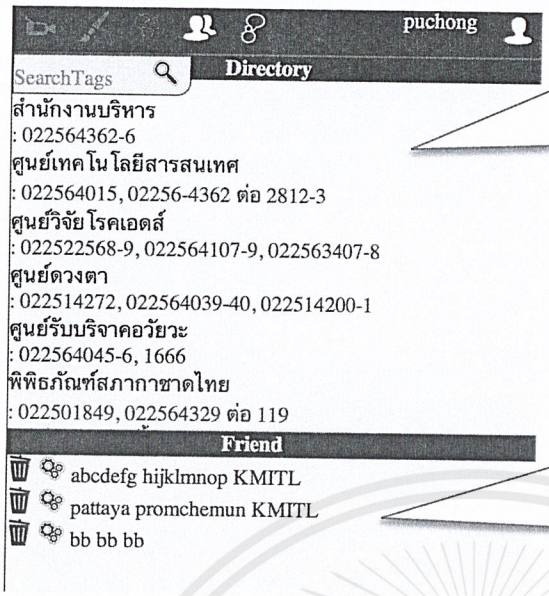
- Videoconference menu



- Whiteboard menu



- Directory contact and friend list



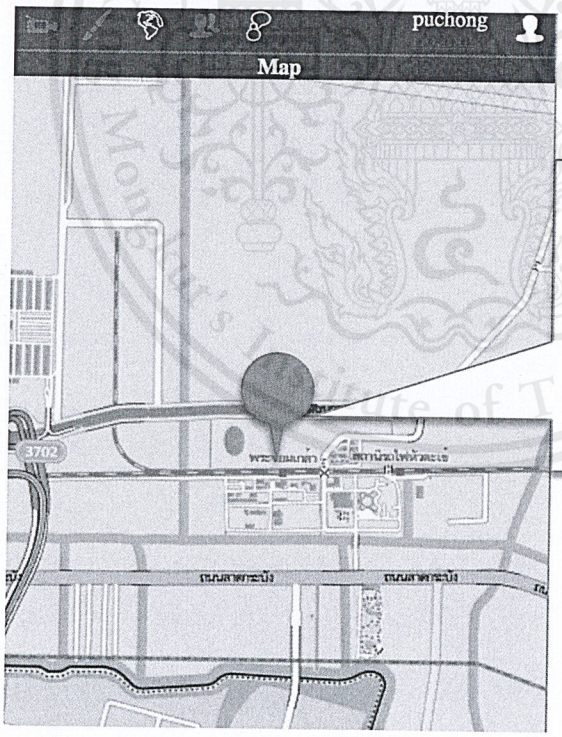
Directory

- User can search for importance organization to communicate with.

Friend list

- User can add/update/delete friend list.
- User can send invitation session to theirs friends.

-Map and Geo-location



Map

- This map will show current position of user and theirs friends.

Chapter 8

Evaluations and Discussions

The main web technology used in this thesis to develop Geocommunicator is the WebRTC. We can now look back to see how WebRTC benefit our development of Geocommunicator and discuss its limitation.

8.1 Benefit of using WebRTC

What we can see as the benefit

- WebRTC is not need to install and update & debug any software or plugins and is updated whenever the browser is updated.
- Working with multi users, across multiple web browsers, across multiple platforms in real-time communication.
- The WebRTC technology provides better sound quality than previously communication.
- Reduced IT costs. A reduction in the complexity system required establishing connections.
- Provide a better emergency service to help saving people's lives.
- Contact directories can access information better than other.

8.2 Limitation of the WebRTC

- WebRTC is new technology only supported in some browsers. The browsers that support the WebRTC consist of Chrome, and Firefox. Other browsers do not support.
- When there are many users in the same session, make video slower, decrease the quality of sound and video, consume bandwidth, and use more computer resources.
- Videoconference is blur and lack when conference more than 6 users, huge bandwidth and CPU-usage out of multiple peer interconnection.
- A low-quality blurred video will be delivered. Then we have to use the **Media-server** to solve this problem.

Chapter 9

Conclusions

9.1 Summary

Our thesis aims to develop novel communication tools using WebRTC and implemented them using JavaScript, Python, Django, and HTML5. The tools are; videoconference, map location, collaborative whiteboard, contact directory, chat, Geo-call. These tools support collaborative communication.

We also design contact directory that can work with GeoCommunicator, and we apply it for emergency rescue application to use by the Thai Red Cross Society. Our contact database is developed using PostgreSQL. The GeoCommunicator can run on a web browser on both a PC and a mobile device.

9.2 Lessons Learned

After we have done the GeoCommunicator project, the knowledge from this project are: The first of all we studied about an object-oriented system is made up of objects that work together to achieve the required functionality, what the system has to do.

Secondly, we analyze the requirement of the system. What are the problems and wishes of users and scope of the proposed system we created use case diagram follow the requirements, which use to specifies what the user wants the system to do. Third, we create some scenario for the use case diagram user wants the system to do. Third, we create some scenario for the use case diagram (Make call, Receive call, Modify contact list, and Find contact) of GeoCommunicator using sequence diagram to show the flow of events.

Next, we design and focus on the class diagram of GeoCommunicator. A class diagram gives an overview of a system by showing its classes and the relationships among them. Fifth, we design ER diagram to store contact. Next, we observed the feature of existing system and design user interface for GeoCommunicator. Afterward we start implemented the code. For the each step of development system, it is useful for software designers (when they are maintaining it, when they are writing code) to achieve the goal of project in time.

Moreover we have studied the ability of the protocols and architectures Real Time Communication Technologies. Understanding the talent of WebRTC is also the most important thing that we have to finish the GeoCommunicator Project. If we have a chance to go back to the past, we will study GeoLocation that use to develop GeoCommunicator project.

9.3 Problems and obstacles

Videoconference is blur and lack when conference more than 6 users, huge bandwidth and CPU-usage out of multiple peers interconnection: To understand it better; assume that 10 users are sharing video in a group. 40 RTP-ports (i.e. streams) will be created for each user. All streams are expected to be flowing concurrently; which causes blur video experience and audio lose/noise (echo) issues.

For each user: 10 RTP ports are opened to send video upward i.e. for outgoing video streams, 10 RTP ports are opened to send audio upward i.e. for outgoing audio streams, 10 RTP ports are opened to receive video i.e. for incoming video streams, 10 RTP ports are opened to receive audio i.e. for incoming audio streams. Maximum bandwidth used by each video RTP port (media-track) is about 1MB; which can be controlled using "b=AS" session description parameter values. In two-way video-only session; each peer uses 2MB bandwidth; otherwise; a low-quality blurred video will be delivered. Then we have to use the media server to solve this problem.

The First time we cannot share screen to another client then we use HTTPS to solve it. It can share theirs own screen window to other clients.

3G-4G problem: any devices that run on cellular network such as 3G or 4G network will have problem black screen video of videoconference. Because of channel video/audio that send to each other is loss. Using TURN Server to solve this problem: TURN is used to relay audio/video/data streaming between peers.

Change Port Signaling server to Port 80. Because the most of devices cannot use specific port of Websocket such as Port: 8080,8000,5060 etc.

9.4 Further work

Our Geocall can easily implement a nearby-friend feature as follows. The users can Geocall to another users by click real time position icon on the map. Find near friend this feather will show who is near user that easier to meeting or conference with them. Step to use nearby-friend feature: 1.login 2.show current location 3. Find friend.

- To improve the performance of the videoconference it by using media-server to handle sessions for multiple users and to record audio and video conversation for the users.

- Adding more functions to the meeting whiteboard so that the user can redo/undo their drawings.

Bibliography

[1] **Getting Started with HTML5 WebSocket Programming**. Vangos Pterneas, Packt Publishing.

[2] **HTML5 Canvas Cookbook**, Eric Rowell, Packt Publishing.

[3] **WebSocket.org** http://www.websocket.org

[4] **Web Platform docs** http://docs.webplatform.org/wiki/apis/websocket/WebSocket

[5] **HTML5 rocks** http://www.html5rocks.com/en/features/connectivity

[6] **HTML5 demos** http://html5demos.com

[7] **Mozilla Developer Network** https://developer.mozilla.org/en-US/docs/WebSockets

[8] **The WebSockets API (W3C)** http://www.w3.org/TR/websockets

[9] http://www.totnetcall.com

[10] http://www.webrtcbook.com/excerpt/excerpt.pdf

[11] https://oncampus.oberlin.edu/webteam/2012/09/architecture-django-templates

[12] http://www.redcross.or.th/aboutus

[13] http://thebuild.com/presentations/unbreaking-django.pdf

[14] https://www.webrtc-experiment.com/video-conferencing