FEBRUARY 25, 2015

DATA COMMUNICATIONS (COMP 4985)

ANDROID GPS

RHEA LAUZON // JEFF BAYNTUN // MICHAEL CHIMICK // JULIAN BRANDRICK
4O

Contents

Requirements

- Android application that allows a smartphone to access a server via TCP/IP
- Must implement the channels between two client devices and the server using TCP/IP
- No constraint on languages or tools
- Must be completed on android and/or iOS

Client Application

- Client acquires its current location
- Client sends the co-ordinates to a receiving server over Wi-Fi using TCP or UDP connections to the server
- Client app prompts the user for an IP address and port number of the server
- After entering the address information of server, app collects location info and sends it to the server periodically

Server Application

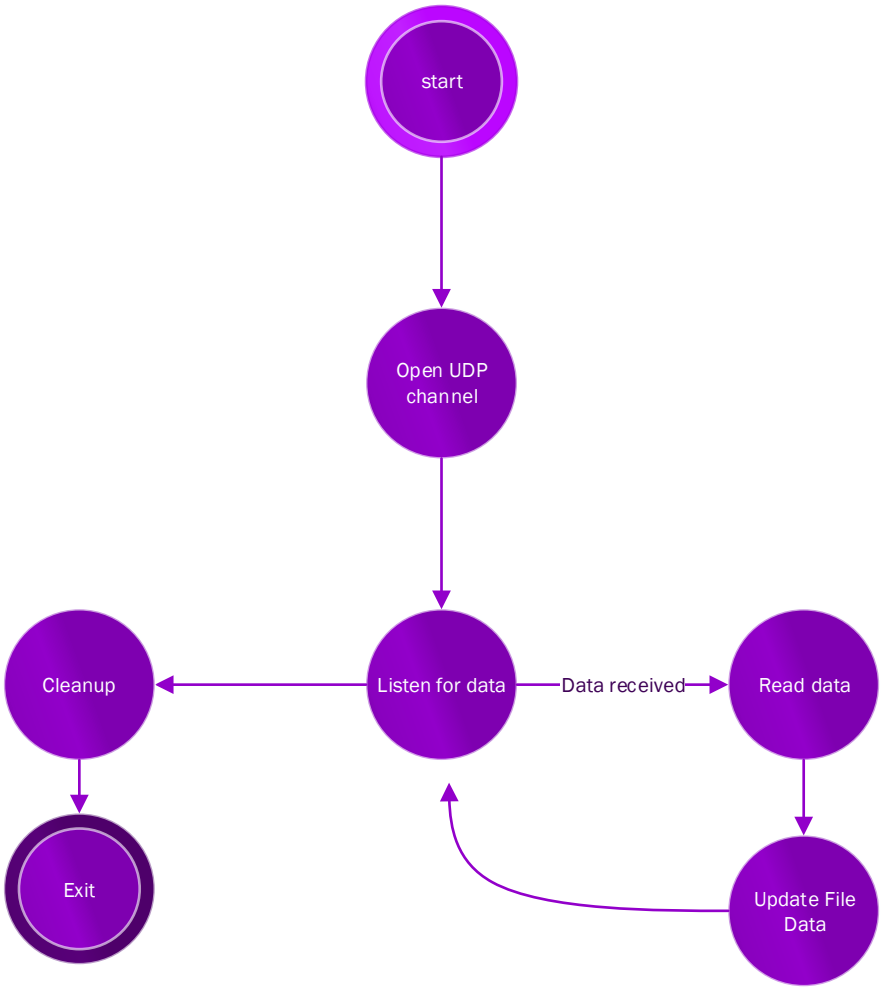
- Runs on Linux machine with apache web server running
- Receives the location data and formats the data in a file
- Data should include:
    - The time the co-ords were received
    - IP address and name of the client device
    - Latitude of device
    - Longitude of device
- Generates a file in the default apache home directory for the web page
- Can receive updates from multiple client devices and generate update files accordingly

Server Web Page

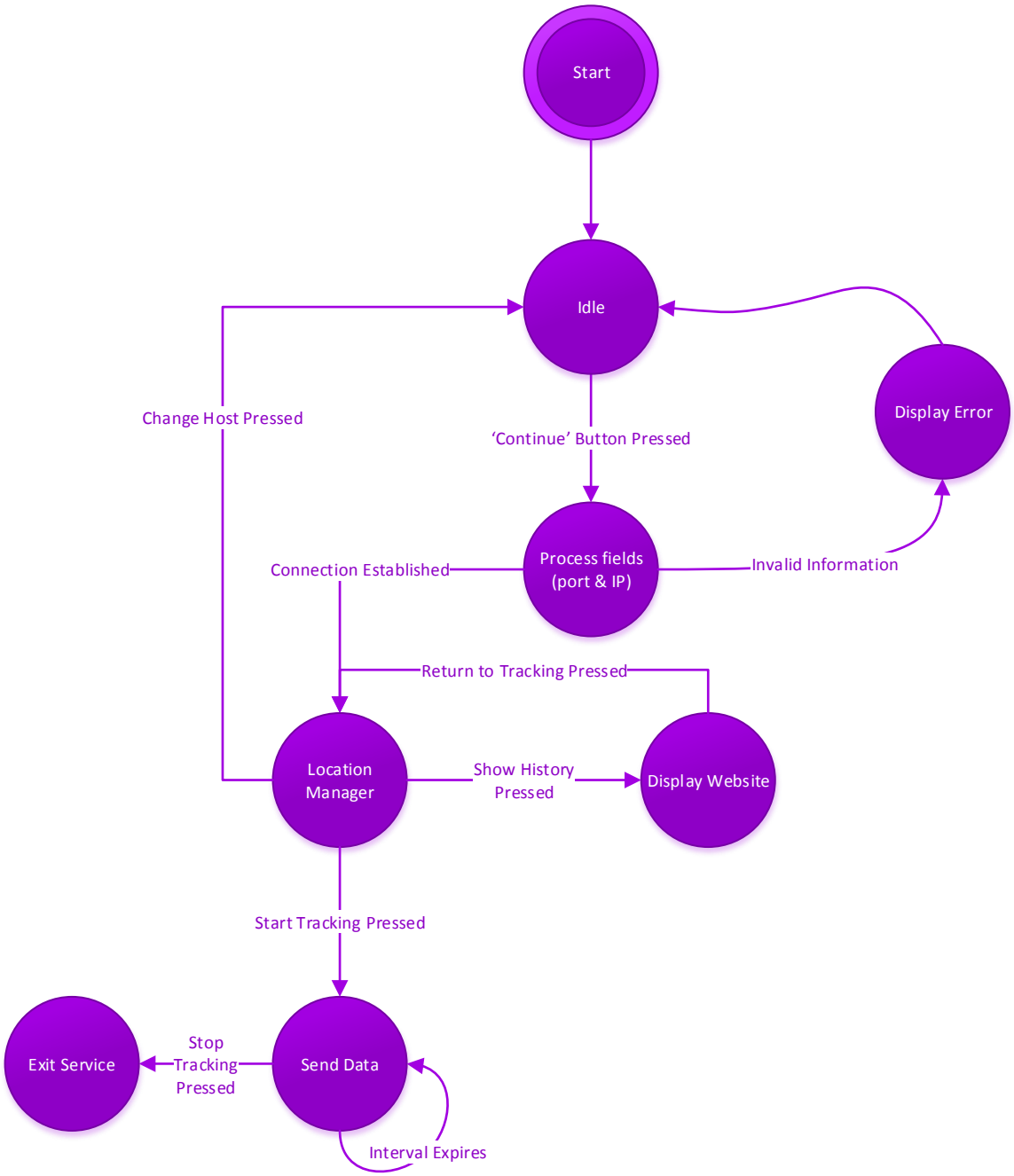
- Server will read the file and plot the co-ords of the clients on a map using google maps API
- Can be viewed remotely using web browser
- ***Must*** have password authentication for access

State Flow Diagrams

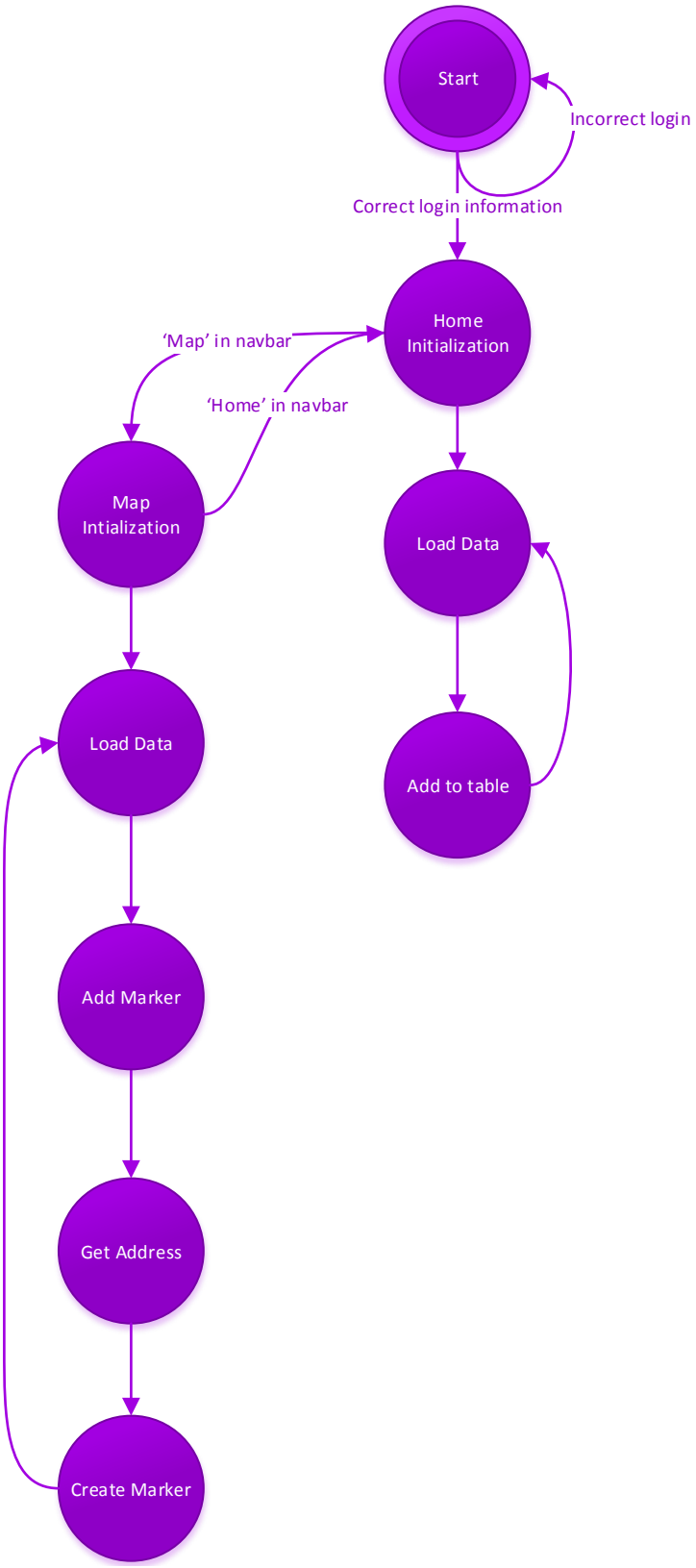
Server Application



Client Application



Server Website



Pseudocode

Website

Home Page

```
home function
{
    when the document is fully loaded
        call the load data function
}


load data function
{
    clear all the data from the table

    set up an http request for the data file
    fetch the data file

    for each user in the XML
        parse the latitude
        parse the longitude
        parse the name
        parse the time
        parse the IP

        call the add to table function

    set a timer to call the load data function every 5 seconds

}



add to table function
{
    append the name, ip, time, latitude,  and longitude
    to the table via html elements

}
```

Map Page

```
ready map function
{
    when the document is full loaded
        call the initialize function
        call the load data function
}

initialize function
{
    create the stylized map via JSON objects
    setup the map options
    create the google map object
    set the style on the map
    initialize the geocoder

    fetch the geolocation of the user running the website
    if their browser does not allow geolocation
        display errror
    else center the map to their location
}


load data function
{
    clear all the markers from the map

    set up an http request for the data file
    fetch the data file

    for each user in the XML
        parse the latitude
        parse the longitude
        parse the name
        parse the time
        parse the IP

        call the add marker function

    set a timer to call the load data function every 5 seconds
}


add marker function
{
```

```
    create the latitude-longitude object based off parameters
    call asynchronous get address function
}

Get Address function
{
   Parse the latitude and longitudes as floats
   Create a latitude-longitude object based off the parsed values
   Use the geocoder to get the physical address of the device
   Call create marker function
}

Create marker function
{
    Make the google maps marker with the location and icon and add it to the map
    Add the name, IP, time, and address to a new info window
    Connect the info window to the marker

    Add a hover listener for mouse over on the marker to display the info window
    Add an on click listener to the marker to zoom the map to the pin

   Add the marker to the list of markers

}
```

Android Application

Start State

```
{
Load the GUI for the main activity
Display inputs for port and host
}
```

Idle State

```
{
wait for user to input port and host information
if both fields have data, change button from red to green
}
```

Process Fields State // invoked by pressing submit

```
{
if port and host do not both have data
	display error message, return to idle
attempt connection with given port and host
if unable to connect
	display error message, return to idle
store port and host for later use
launch Tracker Activity, proceed to Location Manager State
}
```

Location Manager State

```
{
display GUI with options to: start tracking, display the website, change host
if start tracking is pressed
	spawn a background service to handle sending the location data, in Send Data
State
	display Stop Tracking Button
if stop tracking is pressed
	kill the tracking service
	change the button to start tracking
if change host is pressed
	launch Main Activity, goto Idle State
if display website is pressed
	launch Website Activity
	goto Show Website State
}
```

Send Data State

```
{
Start Location Manager
Set Listener to provide data every 15 minutes
while true
       if data provided
       open connection to server
       send data to server
       close connection to server
}
```

Display Website State

```
{
Open browser inside of Current Application using host address provided
display page
show button to return to tracking or toggle website
if return to tracking pressed
       launch Tracking Activity
       goto Location Manager State
}
```

UDP Networking

```
function InitSocket
{
    initialize networking components
        create UDP socket
        pull server address based on host name
}

function SetLocalIP
{
    pull ip address of local device
}

function SetPacketData
{
    set data of UDP packet
}

function Run
{
```

```
    send UDP datagram in separate thread
}

function Teardown
{
    close UDP socket
}

function FoundServer
{
    error-checks to see if the networking setup completed successfully
}
```

Server

```
main
{
    get port from command arguments
    create a server object with the port
    start a server thread
}

run
{
    wait for a packet on the UDP socket
    call add point function with packet data
}

addPoint
{
    find end tag of xml file
    read everything before that tag into buffer 1

    call get point function and assign result to buffer 2

    append buffer 2 to buffer 1
    rewrite xml file
}

getPoint
{
    parse string for the 5 data fields
    return the formatted string for the xml file
}
```