Introduction

I have implemented a simulation of the ringer manager application. The application is coded in `Node.js` and the output can be seen over the command line. I will document the steps to run my program as well as a brief explanation (a detailed flow cum explanation can be seen along comments in my code) of the same and an intermediate report.

Instructions to run my program

1. Install `Node.js`.
2. Run `npm install` at the root directory (`src`) to install all required packages that my application uses.
3. Run `node main.js` to execute my application.
4. The program will print out important simulation metrics on the console, as well as all the relationships in the end along with their weights (which have been calculated after accommodating the feedback at every step).
5. The requirement of course is that the server at http://yangtze.csc.ncsu.edu:9090/csc555/services.jsp should be up and running.

Workflow of my program

1. It parses relationships from **relationships.txt**. This file contains known relationships (mentioned in the specification) and their corresponding relationship types.
2. Then, it parses known places from **places.txt** which also contains the default expected ringer mode there.
3. Next, it parses the relationship types from **relationshipTypes.txt** which contains a contribution factor for each relationship type. This will be later used in my social benefit function.
4. It starts simulating visits.
    1. It goes through all places available from the **places.txt** file.
    2. It enters that place through the web service call.
    3. It lists all the neighbors there.
    4. It requests a call in that location.
    5. Calculates the social benefit fucntion based on the neighbors, the call and the place.
    6. Provides the response and gets back a feedback.
    7. Accommodates the feedback into the contribution factors for each relationship.
    8. Exits the place.

Social benefit function

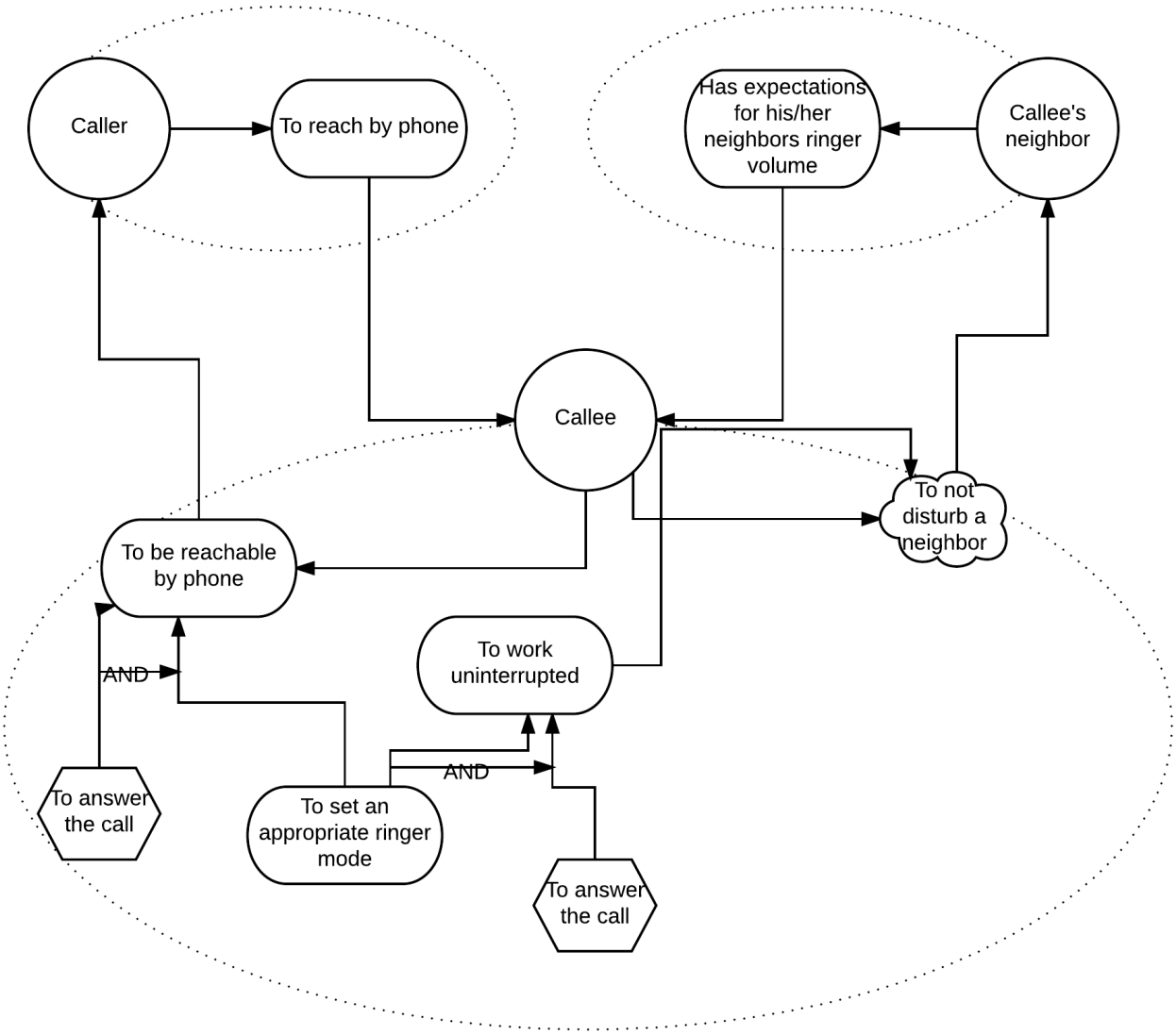
This function has been completely been implemented along with comments in the `main.js` file. For a detailed workflow, please refer to that file. To give a brief overview of what it does:

1. Iterate over all neighbors.
2. Take the neighbor's weight from a map (this starts as 1, but changes based on the feedback received).
3. Multiply the weight with the neighbor relationship type default contribution factor and add to the expected ringer mode property (this starts from 0 for each ringer mode). Thus family contributes more than stranger to deciding the response type.
4. Add a component to the ringer mode property based on the place as well.
5. Find the maximum among the three ringer mode properties. If it is `silent` or `vibrate`, add a component for `vibrate` and `loud` ringer modes respectively based on the urgency of the caller. If the ringer mode chosen is loud, then no need to change it.
6. Find the maximum again and return that response.

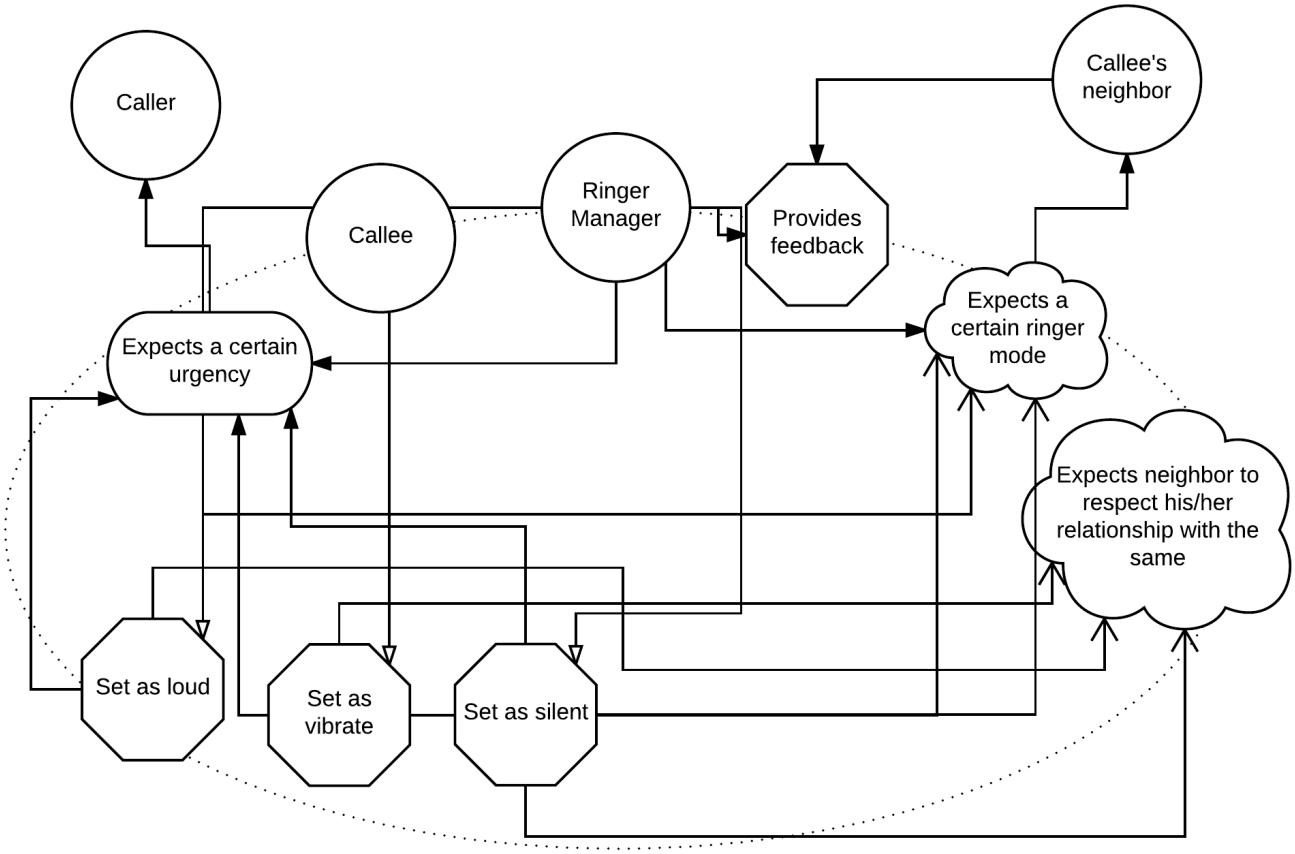
Report

The overall description and workflow of my program and implementation is mentioned in the above parts as well described through my code. As I had depicted befire, almost the whole of the implementation has been done.

The system-as-is model can be noted in the following image. As you can see, it almost remains the same as the one presented in the Xipho paper, with certain modifications. Specifically, as my implementation does not care about returning a message right now, so that part is removed.



The system-to-be changes a little bit more because it incorporates the relationship type contribution on the response decision of the ringer manager, expected ringer mode as well as the feedback from the callee's neighbor and urgency denomination of the caller.


Caller
Callee
Ringer Manager
Provides feedback
Callee's neighbor
Expects a certain urgency
Expects a certain ringer mode
Expects neighbor to respect his/her relationship with the same
Set as loud
Set as vibrate
Set as silent